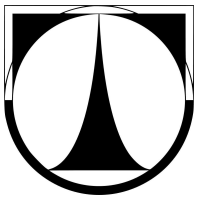
Technická Univerzita v Liberci

Fakulta mechatroniky, informatiky a
mezioborových studií



Bakalářská práce

Liberec 2012 Jan Petřvalský

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Nástroj pro testování bezpečnosti WWW aplikací

Web application security testing tool

Bakalářská práce

Autor: **Jan Petřvalský**
Vedoucí práce: Mgr. Jiří Vraný, Ph.D.

V Liberci 16.5.2012

Před svázáním místo téhle stránky vložíte zadání práce s podpisem děkana (bude to jediný oboustranný list ve Vaší práci) !!!!

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 — školní dílo.

Beru na vědomí, že Technická univerzita v Liberci(TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování

Děkuji Mgr. Jiřímu Vranému, Ph.D. za cenné podněty a odborné konzultace.

Název práce:

Nástroj pro testování bezpečnosti WWW aplikací

Autor: Jan Petřvalský

Obor: Informační technologie

Druh práce: Bakalářská práce

Vedoucí práce: Mgr. Jiří Vraný, Ph.D.
NTI

Abstrakt:

Tato práce popisuje nástroj pro testování bezpečnosti webových aplikací. Nástroj umožňuje otestovat části vyvíjené webové aplikace na Cross Site Scripting, SQL Injection, Clickjacking a zranitelnosti založené na HTTP Response Splitting. V jednoduchém grafickém rozhraní je možné testovat manuálně zadané formuláře, či automaticky otestovat všechny formuláře dostupné na určitém URL. Nalezené problémy a podrobnosti o testech jsou zobrazeny ve stromové struktuktuře s možností exportu výstupní zprávy do hypertextového dokumentu.

Klíčová slova: bezpečnost, webové aplikace, http response splitting, cross site scripting, sql injection, clickjacking

Title:

Web application security testing tool

Author: Jan Petřvalský

Abstract:

This work describes tool for web application security testing. Tool enables test parts of developed web application on Cross Site Scripting, SQL Injection, Clickjacking and vulnerabilities based on HTTP Response Splitting. In simple graphic interface is possible to test manually entered formulars or automatically test all formulars on entered URL. Then issues and test details are displayed in tree structure with possibility to export HTML formatted output report.

Key words: security, web applications, http response splitting, cross site scripting, sql injection, clickjacking

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

SQL	Structured Query Language (Strukturovaný dotazovací jazyk) 17
CR	Carriage Return (Návrat vozíku) 15
LF	Line Feed (Posun o řádek) 15
DOS	Denial of Service(Odepření služby) 12
HTTP	Hypertext Transfer Protocol 13
HTTPS	Hypertext Transfer Protocol Secure 38
URL	Uniform Resource Locator 13
URI	Uniform Resource Identifier 13
XSS	Cross Site Scripting 14
HTML	HyperText Markup Language 15
DOM	Document Object Model 15
OWASP	Open Web Application Security Project 11
PHP	PHP: Hypertext Preprocessor 19
DVWA	Damn Vulnerable Web App 19
WVS	Web Vulnerability Scannner 26
YAML	YAML Ain't Markup Language je formát pro serializaci strukturovaných dat 31
CSRF	Cross Site Request Forgery 12
DAO	Data Access Object 44
SID	Session ID 47
XHTML	Extensible Hypertext Markup Language 43
SEO	Search Engine Optimization 13
TLD	Top Level Domain 16

Kapitola 1

Úvod

Podle statistik za rok 2010 společnosti WhiteHat Security má průměrná webová stránka 230 závažných bezpečnostních nedostatků [16]. Ze zranitelností má nejvyšší zastoupení s 64% Cross Site Scripting. Tato zranitelnost se umisťuje společně se zranitelnostmi SQL Injection a CRLF Injection na předních místech žebříčku, který je každoročně sestavován organizací OWASP[1]. V dnešní době internetových obchodů, elektronického bankovnictví a sociálních sítí získávají na důležitosti zranitelnosti Clickjacking.

Testovat bezpečnost je možné pomocí profesionálních nástrojů pro penetrační testování – vyžadují však hlubší znalosti v problematiky bezpečnosti webových aplikací. Druhou možností je využít automatických nástrojů pro vyhledávání zranitelností. Nevýhodou automatických nástrojů je hlavně vysoká cena a striktní licenční podmínky výrobců.

Tato práce si klade za cíl vytvořit jednoduchý grafický nástroj s otevřeným zdrojovým kódem, který by mohl vývojář webové aplikace použít pro otestování jednotlivých částí vyvíjené aplikace.

Nástroj by měl automaticky vyhodnocovat úspěšnost útoku a výsledky přehledně zobrazovat uživateli.

Teoretické pozadí práce je obsaženo ve druhé kapitole s názvem *Vybrané zranitelnosti webových aplikací*, ve které jsou charakterizovány vybrané zranitelnosti a uvedeny možné způsoby obrany. Obsahem třetí kapitoly nazvané *Srovnání nástrojů pro automatické testování* jsou výsledky testu existujících nástrojů a jejich porovnání. Navržená pravidla pro testování na jednotlivé zranitelnosti, formát pro uložení vektorů a základní části aplikace jsou popsány ve čtvrté kapitole *Návrh nástroje*. V páté kapitole *Implementace nástroje* jsou popsány nejpodstatnější části implementace.

[1] Open Web Application Security Project

Kapitola 2

Vybrané zranitelnosti webových aplikací

V této kapitole jsou popsány vybrané zranitelnosti webových aplikací, které jsou podporované vytvořeným nástrojem. Všechny tyto zranitelnosti patří do skupiny takzvaných chyb validace vstupu spočívajících ve vložení neočekávaného řetězce vytvořeného tak, aby došlo k ovlivnění logiky aplikace.

2.1 Cross Site Request Forgery

2.1.1 O zranitelnosti

Cross Site Request Forgery je útok mířený vůči koncovému uživateli aplikace, který spočívá v provedení požadavku s následným vyvoláním akce v aplikační logice. Od běžného chování se liší ve skutečnosti, že tento požadavek nepochází z legitimního zdroje – je vykonán bez vědomí uživatele, ale s jeho bezpečnostním kontextem. Útočník se tedy snaží uživatele „donutit“ pomocí sociálního inženýrství, škodlivých odkazů či Javascriptu odesílajícího POST formulář, aby požadavek vykonal.

2.1.2 Rozdělení

Útoky CSRF[1] lze rozdělit do dvou druhů – perzistentní (stored) a neperzistentní (reflected). V případě perzistentního CSRF útočník využívá samu cílovou aplikaci k distribuci škodlivého kódu[2]. Útoky zneužívající Stored CSRF zranitelnosti mají vyšší pravděpodobnost úspěchu. Pokud je uživatel schopen přijmout exploit, tak musel být přihlášen a tedy také autorizován provádět akce aplikační logiky dle svých oprávnění. Například pomocí něj lze realizovat DOS[3] vložením kódu provádějícího odhlášení.

Útoky založené na neperzistentním CSRF využívají k distribuci škodlivého kódu externí zdroje. Tím mohou být fóra, komentáře pod články, zprávy „instant messagingu“ a další. Zde

[1] Cross Site Request Forgery

[2] Škodlivým kódem je zde míněn odkaz v případě použití metody GET a kód ve skriptovacím jazyku generující formulář v případě metody POST.

[3] Denial of Service(Odepření služby)

existuje vyšší pravděpodobnost neúspěchu, ale má však pro útočníka zásadní výhodu – celý průběh útoku má plně ve své kontrole. To mu umožňuje zahlazovat za sebou stopy a podstatně tak ztížit pozdější pátrání. Klasickým příkladem často zneužívané aplikace pomocí neperzistentního CSRF jsou webové ankety.

2.1.3 Obrana

Proti CSRF bylo vytvořeno množství funkčních i ne zcela funkčních způsobů obrany, se kterými je nutné počítat.

Dříve bývala uváděna jako obrana kontrola hlavičky HTTP[4] referer. V hlavičce HTTP referer odesílá prohlížeč URI[5] předchozí stránky spolu s požadavkem na danou stránku. Tento způsob obrany nefunguje, poněvadž tuto funkci někteří uživatelé vypínají z důvodu vyššího soukromí a útočníci ji mohou zfalšovat.

Stejně tak není obranou výhradní použití metody POST u formulářů. Útočník by pouze navíc musel vygenerovat formulář a pomocí Javascriptu ho nechat uživatele odeslat.

Další možností je použít přepisování URL[6]. To je samozřejmě vhodné použít z hlediska SEO[7], ale v případě bezpečnosti se jedná o princip Security by obscurity[8]. Útočníkovi nebo penetračnímu testerovi pouze přidělá práci navíc. Avšak s přepisovanými URL mohou mít problém automatické skenovací nástroje, poněvadž musejí používat heuristické metody k identifikaci parametrů.

V současnosti je nejčastěji proti CSRF implementována obrana pomocí tokenů (někdy bývá nazývána metoda podepsaných formulářů). Spočívá v tom, že k odesílaným datům v HTTP požadavku je přidána hodnota nesoucí textovou informaci náhodného charakteru zvaná token. Na serveru je porovnána hodnota tokenu s předem vygenerovanou referenční hodnotou a pokud se rovnají, tak se s nejvyšší pravděpodobností jedná o legitimní požadavek a je zavolána příslušná metoda obchodní logiky. Token může být na server odesílán ve skrytém poli formuláře či v cookie. Hodnotu tokenu je možné vygenerovat při přihlášení a poté ho sdílet mezi všemi formuláři po celou dobu platnosti relace. Bezpečnější variantou je vytvářen nový token pro každý formulář zvlášť. Samotná hodnota může být brána z generátoru náhodných čísel, identifikátoru relace (Session ID) nebo či jejich kombinací za použití hašovacích či kryptografických funkcí.

Referenční hodnota může být uložena v relační proměnné, nebo může být uložena jen na straně klienta, ale na dvojím místě – ve skrytém poli formuláře a v cookie. Druhou možnost lze jednodušeji přidat do stávající aplikace, ale plně spoléhá na správnost implementace Same origin policy[9] v prohlížeči.

2.1.4 Testování a CSRF

Testovat webovou aplikaci na CSRF zranitelnosti automaticky by bylo velice složité, nehledě na možnou destruktivnost akcí. Nástroj by si musel uložit stav aplikace (resp. informace o stavu),

[4]Hypertext Transfer Protocol
[5]Uniform Resource Identifier
[6]Uniform Resource Locator
[7]Search Engine Optimization
[8]Použití utajení k zajištění bezpečnosti.
[9]Same Origin Policy je bezpečnostní koncept pro klientské skriptovací jazyky, který spočívá v povolení přístupu pouze ke zdrojům pocházejícím se stejné domény jako skript [2]

provést operaci a poté nějakým způsobem určit, zda byl útok úspěšný či nikoliv.

I když automatický nástroj není schopen provádění testů na CSRF, tak musí mít implementovánu podporu pro CSRF obranu. Je tedy nutné, aby byl minimálně schopen odesílat skrytá pole, přidávat do HTTP dotazů hlavičky a nastavovat cookies. Dále by měl mít implementovánu podporu pro aktualizaci hlaviček a cookies z HTTP odpovědi serveru. Při použití odlišných CSRF tokenů pro jednotlivé formuláře je nutné pomocí dodatečného dotazu zajistit, aby byla hodnota vygenerována.

2.2 Cross Site Scripting

2.2.1 O zranitelnosti

Útok cross-site-scripting je jednou z nejčastěji se vyskytujících se zranitelností v současných webových aplikacích [14]. I přes tuto skutečnost býval v minulosti považován za metodu začínajících hackerů a jeho závažnost byla vývojáři a administrátory podceňována či znevažována. V poslední době je znát pokrok v boji především proti perzistentnímu XSS[10], které je používáno často k poškození vzhledu napadené stránky. Z principu není útok nijak maskován a je tedy snadné ho nalézt a odstranit.

Zranitelnost XSS se řadí do skupiny chyb validace vstupu. Útok spočívá ve vložení kódu, nejčastěji ve skriptovacím jazyku Javascript místo očekávané obyčejné textové hodnoty. Ten je po zobrazení prohlížečem vykonán v bezpečnostním kontextu napadené domény. Díky tomu je dodržena Same Origin Policy a útočník tak má přístup k důvěrným informacím jako jsou například cookies.

Tímto způsobem je možné provést pestré množství různých útoků od změny vzhledu napadené stránky až po unesení uživatelské relace či krádeže identity. Jedna chyba validace vstupu zákonitě útočníkovi poskytuje další možnosti k průniku do systému, poněvadž pomocí ní může obejít ostatní zabezpečení. Může tak umožnit obejít obranu proti CSRF a útočník následně může být schopen eskalovat svá práva pomocí provedení nelegitimních požadavků.

2.2.2 Rozdělení

I Cross Site Scripting lze rozdělit podle umístění zákeřného kódu na perzistentní a neperzistentní variantu. Perzistentní XSS též zvané Stored XSS je útok směřovaný vůči části logiky webové aplikace, která manipuluje s daty (ukládá a následně je zobrazuje uživateli). S tímto typem útoku se lze setkat převážně u webových fór, komentářích pod články a v neposlední řadě u sociálních sítí.

První krok, který musí útočník ke zneužití XSS zranitelnosti provést, je příprava škodlivého kódu ve skriptovacím jazyce. Následně je spolu s dalšími daty odešle na webový server. Pokud aplikační logika nezkoumá validitu vstupu nebo je filtrování nedostatečné, tak je útočníkův kód uložen do databáze či souboru. Po zobrazení příspěvku prohlížeč, za předpokladu povoleného klientského skriptování, automaticky vykoná škodlivý kód v bezpečnostním kontextu domény.

Závažnost spočívá v tom, že útok je proveden zcela automaticky a bez potřeby ho cílit na

[10] Cross Site Scripting

určitého uživatele. Může však být při konkrétním útoku vyžadována určitá míra spolupráce uživatele (např. kliknutí na tlačítko play u přehrávače).

Nejčastějším cílem při provádění perzistentních XSS je snaha poškodit vzhled napadené stránky (tzv. Website Defacement), využít stránku jako proxy server k provádění dalších útoků, či šířit XSS červy.

U neperzistentní XSS (reflected XSS) není škodlivý kód uložen na cílovém serveru[11], ale je předáván jako součást parametru při požadavku na zobrazení stránky. Webový server z přijatých parametrů vytvoří statický HTML[12] dokument a odešle ho uživateli.

Pokud má prohlížeč povoleno klientské skriptování, tak je kód vykonán v bezpečnostním kontextu domény. Ne vždy musí být útok nutně vykonán, skript může být zablokován rozšířením prohlížeče (např. NoScript) nebo samotným prohlížečem. Chromium má implementovanou obranu proti reflected XSS útokům - nepovoluje vykonání Javascriptového kódu, který byl předán v parametru.

Takto bývají zneužívány vyhledávací formuláře či chybové stránky, u kterých je zpráva předávána v parametru.

U neperzistentního XSS je v určitých případech nutné použít bypass[13], poněvadž hodnota parametru nemusí být vždy umístěna jako prostý text — může být například obsahem atributu alt u obrázku.

Speciálním případem neperzistentního XSS je „DOM[14]-based XSS“, u kterého je zákeřný skript do webové stránky zakomponován na straně klienta, nikoliv na straně serveru.

2.2.3 Obrana

K zabránění útoků XSS vývojáři webových aplikací používají nahrazení řídících symbolů příslušnými entitami. V případě nutnosti přijímat formátovaný text může být použito HTML společně s filtrováním určitých značek (white nebo black listy) či alternativní syntaxe pro formátování, které jsou webovou aplikací převedeny na HTML.

2.3 CRLF Injection

2.3.1 O zranitelnosti

CRLF Injection, též zvaný HTTP Response Splitting, je útok spočívající v injekci znaků pro přechod na další řádek (CR[15], LF[16]) a nového obsahu HTTP odpovědi. Původní odpověď od serveru prohlížeč zahodí a následně je zobrazena podvržená odpověď.

Zranitelné jsou například aplikace, které provádějí přesměrování na základě nefiltrovaných dat od uživatele pomocí hlaviček.

Na základě této zranitelnosti je možné provést Cross User Defacement (nahrazení obsahu stránky) či HTTP Cache Poisoning (nastavení last modified do budoucnosti).

[11] V úvahu nejsou brány logy.
[12] HyperText Markup Language
[13] Řetězec způsobující uzavření elementu.
[14] Document Object Model
[15] Carriage Return (Návrat vozíku)
[16] Line Feed (Posun o řádek)

2.3.2 Obrana

Obranou je filtrovat ze vstupů znaky CR a LF.

2.4 Clickjacking

2.4.1 O zranitelnosti

Clickjacking je metoda, jejíž pomocí může útočník uživatele přimět provést akce, které by nikdy vědomě neprovedl. Takto lze například Clickjacking zneužít k falešnému hlasování, klikání na reklamy či k útoku zvanému Likejacking – nevědomé kliknutí na tlačítko „like“ či „+1“ u sociálních sítí.

Pro provedení tohoto útoku musí útočník provést následující:

- Načtení stránky do iframe
- Nastavení průhlednosti iframe pomocí kaskádových stylů
- Překrytí rámce kromě místa kam má být kliknuto
- Umístění dalšího HTML elementu na místo kliknutí
- Nastavení vlastnosti z-index tohoto elementu tak, aby byl umístěn až za rámcem

Po provedení těchto kroků je útočník připraven vytvořit aplikaci, která uživatele přiměje kliknout na daná místa. Toho může docílit umístěním tlačítka značícího stažení souboru či pomocí hry.

2.4.2 Obrana

Proti útoku Clickjacking se lze bránit použitím kódu zvaného framekiller. Tento kód znemožňuje načtení stránky uvnitř rámce frame nebo iframe (Kód 2.1). Funkčnost tohoto řešení však závisí na zapnutém Javascriptu. Pokud uživatel má zapnuté rozšíření blokující skripty, tak obrana fungovat nebude. Dále je možné framekiller obejít pomocí Javascriptu či vhodného vstupu [15].

Listing 2.1: Framekiller

```
<script type="text/javascript">
    if(top != self) top.location.replace(location);
</script>
```

Jako obrana proti Clickjacking byla zavedena HTTP hlavička X-Frame-Options. Pomocí této hlavičky může webová aplikace specifikovat chování prohlížeče při načítání stránky v rámcích na externích serverech.

Při nastavení hlavičky na hodnotu „ALLOW-FROM“ je povoleno zobrazení v rámci pro jednu doménu. Hodnota „SAMEORIGIN“ umožňuje vkládání do rámců v téže doméně. Použití této hodnoty však není podle [13] doporučeno, poněvadž stačí shoda TLD[17]. Pro úplné zakázání zobrazení v rámcích slouží hodnota „DENY“.

[17] Top Level Domain

2.5 SQL Injection

2.5.1 O zranitelnosti

Zranitelnost SQL Injection kvůli závažnosti přetrvává na předních místech nebezpečnosti po několik let – v žebříčku OWASP 2010 se Injection flaws umístily na prvním místě. V dnešní době význam ještě stoupá, poněvadž díky rozšíření HTML5 (WebSQL a Indexed Database API) mohou být ohroženi i samotní uživatelé prostřednictvím webového prohlížeče.

SQL Injection je technika spadající do takzvaných chyb validace vstupu, u kterých útočník zneužívá slabého filtrování vstupů systému. Útočník vkládá řídící znaky SQL[18] k provedení bypassu, kterým ukončí původní dotaz a následně přidá vlastní kód provádějící útok. Tyto symboly jsou závislé na databázovém serveru – nejčastěji se jedná o apostrof a dvojitou pomlčku.

Takto může být pomocí SQL Injection vhodným kódem provedeno přihlášení bez znalosti hesla, eskalace práv, DOS útok či krádež dat.

2.5.2 Obrana

Nejlepším způsobem obrany proti SQL Injection je použití parametrických dotazů namísto vytváření dotazu sčítáním řetězců. Parametrické dotazy jsou podporovány všemi hlavními databázovými servery (MySQL, Oracle, DB2, SQL Server či PostgreSQL). Taktéž jsou podporovány v programovacích a skriptovacích jazycích.

Pokud z nějakého důvodu není možné použít parametrických dotazů, tak je možné filtrovat nebezpečné znaky ze vstupů, či escapovat speciální symboly (například funkcí addslashes).

[18] Structured Query Language (Strukturovaný dotazovací jazyk)

Kapitola 3

Srovnání nástrojů pro automatické testování

V této kapitole jsou popsány výsledky otestování nástrojů pro vyhledávání zranitelností ve webových aplikacích. Vybrány byly nástroje, které mají zdarma dostupnou variantu, grafické uživatelské rozhraní a umožňují automaticky provádět testy.

3.1 Existující nástroje

Cílem této práce bylo vytvořit nástroj, který by byl snadno použitelný a nevyžadoval hlubší znalosti problematiky informační bezpečnosti či metodik penetračního testování. Takovýto nástroj by mohl využívat samotný vývojář pro testování jednotlivých částí vyvíjené webové aplikace bez toho, aby byl zatěžován samotným procesem testování a mohl se soustředit přímo na řešení eventuálních problémů. Z tohoto důvodu byly vybrány nástroje, které umožňují automatické testování bezpečnosti.

Pro otestování byly vybrány komerční nástroje ve zdarma dostupných verzích. Zástupcem opensource je zde Gamja a OWASP Zed Attack Proxy – tento nástroj je však aplikační proxy server a automatické skenování je pouze dodatečnou funkcí. Gamja postrádá grafické rozhraní.

Testované nástroje jsou společně s verzí a licencí vypsány v tabulce 3.1.

Tabulka 3.1: Testované nástroje

Název	Verze	Edice
Acunetix Web Vulnerability Scanner	8.0 build 20120215	Free
Mavituna Security Netsparker	1.7.2.13	Community
N-stalker Web Application Security Scanner	2012 build 7.1.1.117	Free
WebCruiser Web Vulnerability Scanner	2.5.0	Professional
Websecurify scanner	0.9	Basic
OWASP Zed Attack Proxy	1.3.4	Apache License 2
Gamja	1.6	GNU GPL

3.2 Test nástrojů

3.3 Cílová aplikace a její nastavení

První test (Tabulka 3.2) byl proveden na http://testphp.vulnweb.com:80/, což je vzorová aplikace společnosti Acunetix.

Tabulka 3.2: Výsledky nástrojů při testu Acunetix Vulnweb

název nástroje	XSS	SQLI	omezení	cena
Acunetix Web Vulnerability Scanner	13	7	jen XSS	až $9995
Mavituna Security Netsparker	8	4(5)	méně vektorů, dodatečné funkce	$1950 (Standard) $5950 (Professional)
N-stalker Web Application Security Scanner	7	N/A	jen XSS	až $3199
WebCruiser Web Vulnerability Scanner	5	4	žádná	$49 (Personal) $890 (Enterprise)
Websecurify scanner	0	4	basic engine (více informací není k dispozici)	239,99 USD
OWASP Zed Attack Proxy	6	4	žádná	zdarma
Gamja	2	3	žádná	zdarma

Při druhém testu (Tabulka 3.3) byly všechny nástroje byly otestovány na aplikaci Damn Vulnerable Web App. Tato webová aplikace je napsána ve skriptovacím jazyce PHP[1] a využívá databázi MySQL.

DVWA[2] používá cookie „PHPSESSID“ k autentizaci a cookie „security“ k nastavení úrovni zranitelnosti aplikace. Srovnávány byly nástoje ve zdarma dostupných verzích, které mají určitá omezení ve funkčnosti. Bohužel mezi chybějícími funkcemi bylo právě použití přihlašovacích údajů a nastavení cookies. Aby tedy bylo možné provést testy všemi nástroji, tak byly na aplikaci provedeny následující úpravy:

- Nastavení implicitní úrovně bezpečnosti na low ve skriptu „dvwa
  includes
  dvwaPage.inc.php“.
- Povolení přístupu k aplikaci pro nepřihlášení uživatele zakomentováním implementace funkce „dvwaPageStartup($pActions)“.
- Odstranění nutnosti potřeby „security“ cookie pomocí pevného nastavení proměnné „$vulnerabilityFile“ ve skriptech útoků.

[1] PHP: Hypertext Preprocessor
[2] Damn Vulnerable Web App

Tabulka 3.3: Výsledky nástrojů při testu DVWA

název nástroje	XSS	SQLI	omezení	cena
Acunetix Web Vulnerability Scanner	2	N/A	jen XSS	až $9995
Mavituna Security Netsparker	0	N/A	méně vektorů, dodatečné funkce	$1950 (Standard) $5950 (Professional)
N-stalker Web Application Security Scanner	1	N/A	jen XSS	až $3199
WebCruiser Web Vulnerability Scanner	2	2	žádná	$49 (Personal) $890 (Enterprise)
Websecurify scanner	5	2	basic engine (více informací není k dispozici)	239,99 USD
OWASP Zed Attack Proxy	0	2	žádná	zdarma
Gamja	N/A	N/A	žádná	zdarma

3.4 Zhodnocení nástrojů

Většina komerčně dostupných nástrojů pro testování bezpečnosti webových aplikací je sice nabízena i v takzvané komunitní či verzi zdarma. Jejich použití však není možné nejen kvůli licenčním, ale i funkčním omezením. Jedná se tedy pouze o jinak pojmenované demoverze.

Všechny z testovaných nástrojů byly schopny odhalit únik potenciálně zneužitelných informací (emaily, hesla), zálohy zdrojových kódů či skripty umožňující nahrávání souborů na server.

Vzhledem k omezením nelze z výsledků objektivně určit kvalitu jednotlivých produktů, pouze lze poukázat na jejich určité nedostatky. Za hlavní nedostatek lze označit absenci možnosti editovat stávající a přidávat nové testovací vektory. Nové vektory jsou však zákazníkům nabízeny formou aktualizací.

Uživatelská rozhraní aplikací mají společný prvek – pro výpis nalezených zranitelností používají strom. Tento přístup je mnohem přehlednější než prostý list nebo tabulka.

Nástroje jsou vhodné hlavně pro provádění automatických testů, testování jednotlivých stránek či formulářů je komplikovanější.

Dále umožňují nastavit různé parametry testů jako je adresa proxy serveru, počet vláken či přihlašovací údaje.

Dominantní společnosti (Acunetix, Mavituna Security, N-Stalker) nabízejí své nástroje s velkým množstvím více či méně omezených licencí. Bohužel všechny mají nastavenou cenovou politiku v rozmezí $1000 až $10000. Nejlevnější roční licence však mají omezení pouze na jednu stránku, takže je jejich použití pro vývojáře nevhodné.

Opensource nástroje jsou vhodné spíše pro penetračního testera, vyžadují vyšší znalosti bezpečnosti a její použití je komplikovanější.

Kapitola 4

Návrh nástroje

V této kapitole jsou definovány požadované funkce nástroje, popsán navržený formát pro uložení testovacích vektorů do souboru a navržené struktury pro programovou reprezentaci vektoru, testovaných formulářů a výsledků testů.

Dále jsou popsána navržená pravidla pro automatické vyhodnocování zranitelností s ohledem na možnou přítomnost obrany proti CSRF a možnost perzistentních variant útoků.

4.1 Požadované funkce

Nástroj by měl umožňovat otestovat jeden či více formulářů. V případě přítomné CSRF obrany pomocí tokenů by měl být test cíle taktéž proveden. Formulář by mělo být možné zadat manuálně z textových polí, nebo ho získat automaticky ze zadaného URL.

Výsledek testu by měl být uživateli prezentován v přehledném formátu s možností zobrazení podrobností o provedeném testu – použitý testovací vektor, možnosti obrany, cílová platforma a podrobnosti o zranitelném formuláři. Výsledky by mělo být možné filtrovat zadaním klíčových slov podle typu zranitelnosti, URL a zranitelného parametru. Z výsledků testu by mělo být možné vygenerovat výstupní zprávu formátovanou pomocí HTML.

Hodnoty testovacích vektorů, kterými jsou testy prováděny, by měly být načítány z textového souboru.

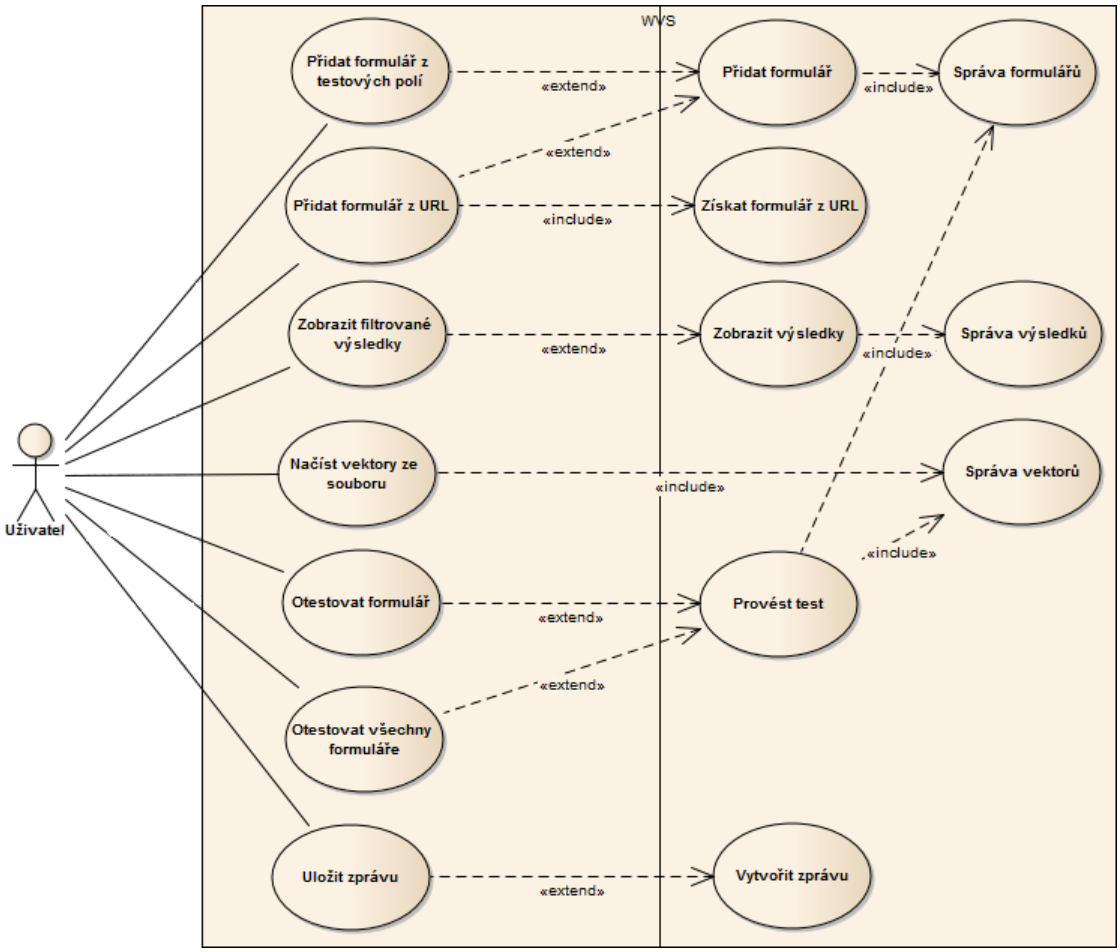
Dále by měl nástroj podporovat komunikaci přes proxy server a mít možnost nastavit kódování.

Funkcionalita je znázorněna v diagramu 4.1.

4.2 Testování bezpečnosti

4.2.1 Black-box metody testování bezpečnosti

K testování bezpečnosti webových aplikací slouží metoda zvaná „fuzzzing“. Tato technika spočívá ve vkládání neplatných, nečekaných nebo náhodných dat do vstupů aplikace. Následně jsou zkoumány výstupy a jsou zaznamenávána všechna vývojářem neočekávaná chování. Při automatickém vyhodnocování úspěšnosti jsou v odpovědi serveru vyhledávány charakteristiky ukazující na úspěšnost útoku.



Obrázek 4.1: Diagram případů použití

Existují dvě metody black-box testování – recursive fuzzing a replacive fuzzing. Rekurzivní fuzzing spočívá v postupném vytváření parametrů HTTP dotazu pomocí iterování napříč všemi přípustnými „kombinacemi“ prvků dané množiny. Toto řešení není v praxi použitelné — zkoušením všech možností je odesíláno ohromné množství HTTP požadavků. Vzhledem k časové prodlevě mezi odeslání dotazu a přijmutím odpovědi je téměř nemožné všechny „kombinace“ projít v přijatelném čase.

V metodě replacive fuzzing je využívána sada předem vytvořených pravidel, které jsou postupně odesílány jako hodnoty parametrů formuláře a následně je vyhodnocován dokument vygenerovaný testovanou aplikací. Tato pravidla jsou nazývána fuzz vektory. Obecně celkový počet prováděných požadavků závisí na počtu pravidel v sadě, ale při praktickém testování může být z různých důvodů (například CSRF obrana) nutné odeslat více požadavků.

Fuzz vektor se obvykle skládá ze dvou částí. První částí je samotný testovací kód pro detekci XSS zranitelnosti, například javascriptová funkce alert(), a druhou částí je takzvaný „bypass“ uzavírající případné HTML značky.

Metoda replacive fuzzing je vhodná pro automatické nástroje a proto byla použita i v tomto nástroji.

4.2.2 Problém perzistentních útoků

U komerčních nástrojů se úspěšnost detekce běžných neperzistentních útoků pohybuje okolo 50% (pro detekci známých zranitelností pomocí vektorů). Velice problematická je však detekce perzistentních útoků – průměrná schopnost detekce perzistentního XSS se pohybuje okolo 15%. Detekce perzistentního SQL Injection dokonce nebyl schopen žádný [3].

V určitých případech je možné detekovat perzistentní XSS za pomocí vyhodnocovací logiky pro běžnou neperzistentní variantu. Například je takto odhalena zranitelnost v případě, kdy je po odeslání uložena hodnota požadavku do databáze a následně je v HTTP odpovědi přijat výpis dat obsahující provedený test.

Návrh bylo nutné přizpůsobit možnosti perzistentních variant útoků. Nejhorší z možností by bylo pro každý vektor uložený v souboru vytvořit dvě programové reprezentace vektoru, instance dvou různých tříd – pro perzistentní a neperzistentní test. Tento přístup je však nepružný a nedodržoval by logickou strukturu – vektor je jeden, ale zranitelnost může mít dva typy.

Další možností je nechat vyhodnocování neperzistentních útoků na třídě vektoru. Hledání perzistentních zranitelností by měla na starosti samotná třída provádějící testování – na všech URL hledat značky neúspěšných testů (všechny nutné informace je možné získat z výsledků).

Poslední možností je použít třídy vektorů jen jako model pro uložení dat a vytvořit modulární třídu provádějící testování. Pro každý typ vektoru by byly dva moduly (pro perzistentní a neperzistentní), které by prováděly testování. Třída skenru by měla na starosti ještě výběr správného modulu na základě typu vektoru.

4.3 Návrh testů

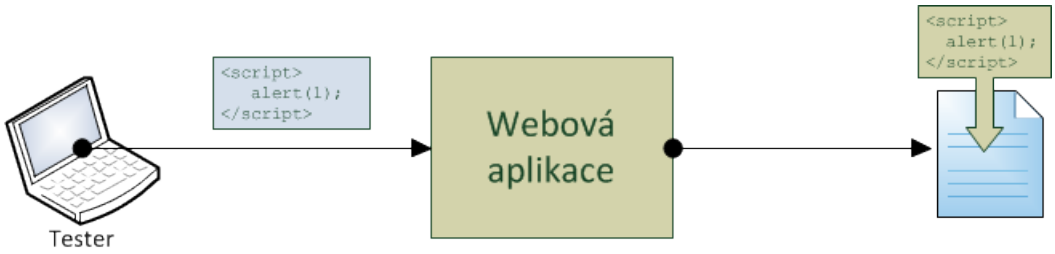
4.3.1 Test na Cross Site Scripting

Při testování na zranitelnost XSS je na vstup předán vektor. Ten je odeslán v parametru jako součást HTTP požadavku testované aplikaci. Webová aplikace provede svojí logiku na základě vstupů a zpět odešle HTTP odpověď. Pokud je parametr a příslušný modul (například PHP skript) zranitelný, tak je součástí odpovědi hodnota vektoru. V případě nezranitelnosti hodnota vektoru v odpovědi chybí, nebo je přítomna jeho filtrovaná varianta – mohou být odstraněny řídící znaky, či být nahrazeny za příslušné entity.

Aby bylo možné po provedení testu vyhodnotit zranitelnost jednotlivých parametrů, tak je nutné v každém požadavku odesílat hodnotu vektoru pouze jedenkrát. Ostatní parametry musejí být nastaveny na neutrální hodnotu (nějaký řetězec). Tento způsob je znázorněn na obrázku 4.2.

U zranitelností XSS je možné odeslat požadavek s nastavenými všemi parametry na testovací hodnotu. Hodnota v každém z parametrů však musí mít odlišný identifikátor, aby by bylo možné je následně v odpovědi serveru odlišit. Problém by však mohl nastat v případě chybového stavu aplikace, poněvadž by nemuselo být možné určit, jaký parametr způsobil chybu. Z tohoto důvodu byla vybrána první možnost.

Při použití obrany proti CSRF na bázi tokenů bude generováno dvakrát více požadavků než bez použití CSRF. Nejdříve je nutné získat aktuální hodnotu tokenu ve skrytém parametru testovaného formuláře a hodnotu tokenu v cookie. Tento přístup je neobecnější – simuluje chování webového prohlížeče a proto byl také vybrán.



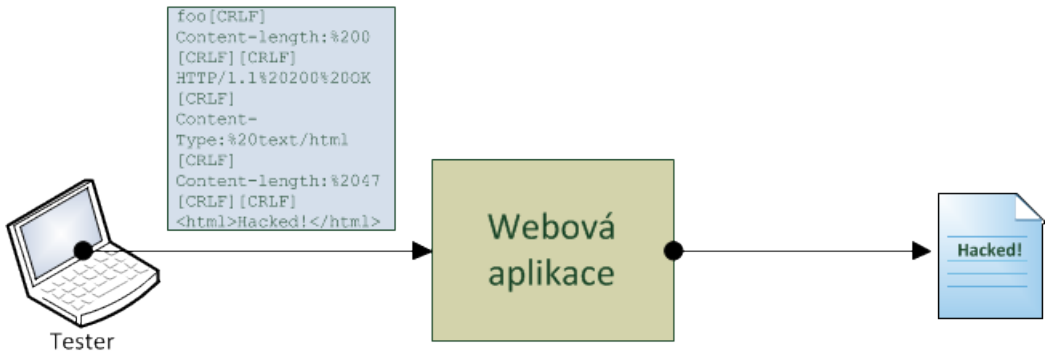
Obrázek 4.2: Testování aplikace na Cross Site Scripting

V případě znalosti implementace CSRF obrany testované aplikace by bylo možné požadavky navíc zcela eliminovat – v případě obrany na bázi porovnávání tokenu z cookie a tokenu ze skrytého parametru by mohly být tokeny nastaveny na nějakou hodnotu. Avšak tento způsob není univerzální a selhal by například při porovnávání tokenu ze skrytého parametru a z relační proměnné.

4.3.2 Test na HTTP Response Splitting

Testování na HTTP Response Splitting funguje podobně jako testování na XSS. Bohužel nelze otestovat více parametrů v jednom dotazu – byl by detekován pouze první zranitelný parametr a ostatní by zůstaly neodhaleny. A to i v případě použití unikátních značek pro jednotlivé parametry.

Vyhodnocení zranitelnosti je triviální. V případě zranitelného parametru by se v těle odpovědi nacházela pouze příslušná značka a případná další část vektoru. Otestování jednoho parametru je znázorněno na obrázku 4.3.



Obrázek 4.3: Testování aplikace na HTTP Response Splitting

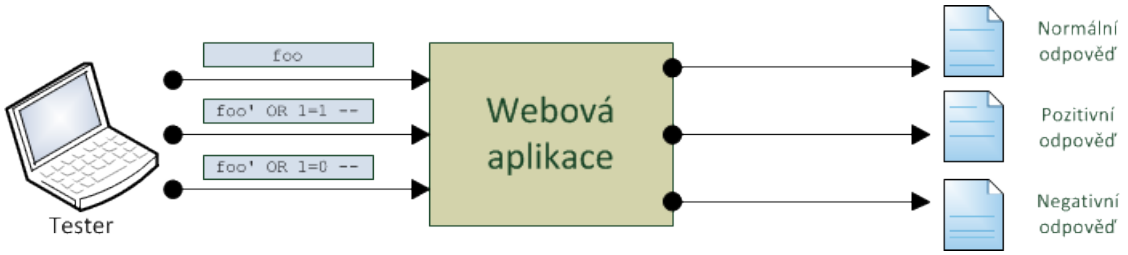
Opět platí, že při použití obrany proti CSRF na bázi tokenů bude generováno dvakrát více požadavků. Nejdříve je nutné získat aktuální hodnotu tokenu ve skrytém parametru testovaného formuláře a hodnotu tokenu v cookie.

Takto lze testovat na všechny útoky založení na HTTP Response Splitting (Cross User Defacement, Cache Poisoning).

4.3.3 Test na SQL Injection

K testování na SQL Injection je použita metoda „Statement Injection“ nazývaná též „Boolean SQL Injection“. Pro automatické testování je upravena tak, že jsou postupně odeslány tři dotazy

a následně jsou porovnávány jejich výstupy. První dotaz, zde označen jako neutrální, je odeslán bez dodatečných úprav a funguje tedy podle úmyslu vývojáře testované aplikace. Následně jsou odeslány dva zmanipulované dotazy. Pozitivní dotaz je zmanipulován tak, aby v případě zranitelnosti byl výsledný SQL dotaz vyhodnocen vždy jako logická pravda. Oproti tomu negativní dotaz je zmanipulován tak, aby končil jako logická nepravda. Otestování jednoho parametru je znázorněno na obrázku 4.4 (bez CSRF podpory).



Obrázek 4.4: Testování aplikace na SQL Injection

Výsledky dotazů jsou uloženy a následně jsou hledány podobnosti. Je předpokládáno, že se úspěšný požadavek od neúspěšného „hodně“ liší. Pokud je si více podobná dvojice pozitivní-neutrální výstup, než dvojice negativní-neutrální, tak je cíl vyhodnocen jako zranitelný. Tento test stojí na myšlence, že nezmanipulovaný dotaz končí například hláškou „neplatné heslo“. Stejně je ukončen i negativní dotaz (vždy je nepravda). Ale pokud je zadáno správné heslo, tak je uživatel přihlášen. Stejně tak končí v případě zranitelnosti i pozitivní dotaz.

Tato metoda má omezení při použití uložených procedur či funkcí, ale i přes to je používána v automatických nástrojích. Z důvodu možnosti automatického vyhodnocování, nezávislosti na implementaci cíle a nezávislosti na platformě byla tato metoda použita v této práci.

Opět není možné rozlišit jednotlivé parametry při provedení jednoho hromadného dotazu, proto musejí být otestovány všechny parametry zvlášť.

Při testování formulářů chráněných pomocí CSRF tokenů je nutné odeslat pro každý požadavek z trojice navíc ještě dotaz pro nastavení hodnoty tokenu. Celkem je tak odesláno $2 \cdot (3 \cdot n)$ dotazů, kde n je počet parametrů.

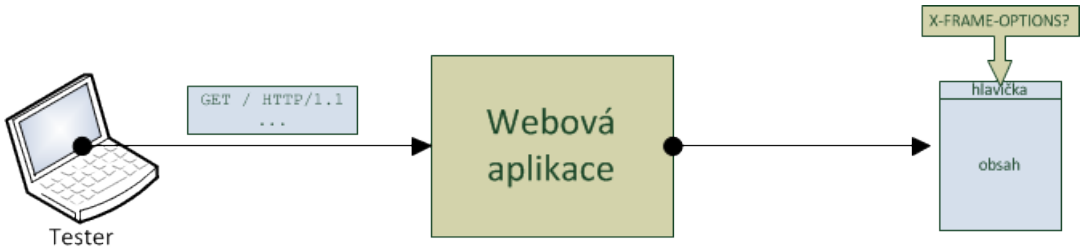
4.3.4 Test na Clickjacking

Útok Clickjacking není testován pomocí odesílání testovací hodnoty v parametrech na webový server, ale je prováděna analýza hlaviček. Pro otestování je potřeba provést:

- Odeslání HTTP požadavku (i v rámci jiného testu)
- Přijmutí HTTP odpovědi
- Hledání hlavičky X-Frame-Options

Pokud hlavička X-Frame-Options není v odpovědi přítomna, nebo má jinou hodnotu než DENY, tak je cíl vyhodnocen jako zranitelný.

Tento způsob však netestuje obranu založenou na framekilleru.



Obrázek 4.5: Testování aplikace na Clickjacking

4.4 Formát vektoru

4.4.1 Formát a existující nástroje

V dnešní době neexistuje jednotný formát pro ukládání bezpečnostních vektorů. Komerční i open source aplikace používají zcela odlišné proprietární formáty, často využívají svou vlastní databázi. Pokud už program má možnost definovat vlastní vektor, tak je potřeba použít vlastní specializovaný editor. Takovéto řešení používá například Acunetix WVS[1].

OWASP Webscarab a další nástroje pro penetrační testování obsahují modul fuzzer. Ten je schopen načítat vektory z jednoduché textové podoby – vždy jeden vektor na řádku. Použití takto jednoduchého formátu však pro vyvíjený nástroj nebylo vhodné, poněvadž neumožňuje uložení různých druhů vektorů do jednoho souboru a nesplňuje požadavky na uložení dodatečných informací – návrh obrany a cílovou platformu by pro každý vektor bylo nutné uložit do dalšího souboru.

4.4.2 Definice formátu

Protože neexistuje žádný standard pro uložení vektorů, tak byl vytvořen nový formát. Hlavní důraz byl u formátu kladen na snadné čtení a možnost manuální editace uživatelem pouze za použití standardního textového editoru.

Definice nového vektoru je uzavřena mezi klíčové slovo „vector“ a prázdný řádek. Ukončení vektoru prázdným řádkem je povinné. Samozřejmě by bylo možné povolit ukončení klíčovým slovem „vector“ následujícího vektoru, avšak formát by tím ztrácel na přehlednosti.

Navíc soubor je též ukončen prázdným řádkem. Proto za poslední vlastností posledního vektoru musejí být vloženy dva prázdné řádky – jeden ukončuje vektor a druhý soubor. Příklad souboru je uveden ve výpise 4.1. Prázdné řádky jsou zde označeny symbolem „[LF]“.

Listing 4.1: Příklad souboru s XSS a SQLI vektorem

```
1 vector
2     name=Simple XSS
3     type=XSS
4     platform=Vsechny prohlizece
5     value=<script>alert(%s);</script>
6     defense=Z uzivatelskych vstupu filtrovat symboly pro HTML < a >.
7 [LF]
```

[1] Web Vulnerability Scannner

```
vector
    name=Simple SQLI 1
    type=SQLI
    platform=Vsechny databazove servery
    value=' OR 1=1 --
    value=' OR 1=0 --
    defense=Z uzivatelskych vstupu filtrovat apostrof.
    defense=Zavest parametricke SQL dotazy.
[LF]
[LF]
```

Každá vlastnost je zapisována na samostatný řádek. Pro vyšší přehlednost je možné použít odsazení jedním či více bílými znaky – doporučován je jeden „tabelátor“. Název vlastnosti a hodnota je oddělena symbolem rovnosti (přiřazení v notaci jazyku C), který může sousedit z obou stran s mezerou.

Ve formátu je též definováno pořadí (Tabulka 4.1), ve kterém musejí být jednotlivé vlastnosti v souboru zapsány – dle tabulky začíná pořadí polem „name“.

V souboru mohou být uloženy i vektory, u kterých není nutné mít definovány určité hodnoty. Z tohoto důvodu jsou povinné pouze vlastnosti „name“ a „type“. Pokud danou vlastnost vektor nepotřebuje, tak je možné ji zcela vynechat. Alternativně je možné nechat pouze část „vlastnost=“. Významy jednotlivých atributů jsou uvedeny v tabulce 4.1.

Tabulka 4.1: Vlastnosti vektoru

název atributu	počet	význam	povinný
name	1	název vektoru	ano
type	1	typ vektoru	ano
platform	N	zranitelné platformy oddělené čárkou	ne
value	N	testovací hodnota	ne
defense	N	popis možnosti obrany	ne

Podle hodnoty vlastnosti „type“ určuje aplikace konkrétní třídu vektoru jejíž instanci má vytvořit. Tato hodnota také úzce souvisí s počtem polí „value“, které může vektor mít. Cross Site Scripting vektor používá pouze jednu hodnotu, ale například klasický SQL Injection (Boolean SQL Injection) vektor z důvodu vyhodnocování musí obsahovat hodnoty dvě.

Ostatní hodnoty jsou nepovinné a to i s hodnotou value. Ve value sice je uložena testovací hodnota, avšak testy na určité typy útoků (např. Clickjacking) vyžadují analýzu hlaviček, kterou nelze realizovat touto formou. Objekt vektoru pak slouží pouze jako model pro uložení podrobností o útoku.

Vlastnost „platform“ obsahuje výčet systémů, proti kterým může být tento vektor s úspěchem použit. Jednotlivé položky výčtu jsou odděleny čárkou. Pokud se výčet nevejde na jeden řádek, tak je nutné další část zase uvést pomocí „platform=“. Například u XSS vektoru zde budou vyplněny jména a verze prohlížečů.

Každý dobrý vektor by měl uživateli nástroje dát instrukce, jak se proti němu bránit. Tato nápověda je obsažena v polích defense.

4.5 Reprezentace vektorů

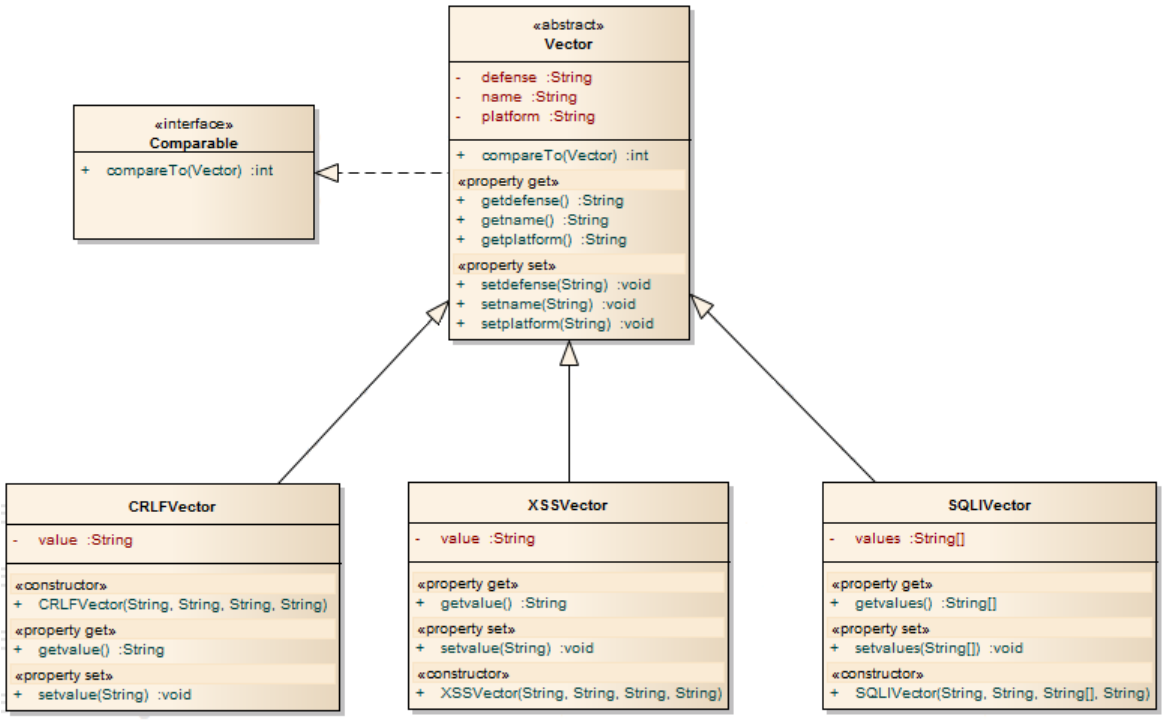
V programu je každý vektor reprezentován instancí třídy dáné typem zranitelnosti, na kterou vektor testuje.

Od abstraktní třídy Vector dědí všechny konkrétní vektory. Vector obsahuje vlastnosti společné pro všechny typy vektorů, tedy název, cílovou platformu a popis obrany. Poněvadž počet testovacích hodnot je závislý na implementaci metody pro vyhodnocování, tak je vlastnost value deklarována až ve třídě konkrétního vektoru.

Atribut platform slouží k uložení zranitelných systémů konkrétním vektorem – jedná se o seznam čárkou oddělených hodnot. Jednotlivé řádky jsou odděleny pomocí znaku pro nový řádek „\n“.

Popis obrany proti danému vektoru je obsažen v atributu defense. Oddělení jednotlivých řádků je stejné jako u atributu platform.

Aby bylo možné jednotlivé vektory porovnávat, tak třída Vector implementuje rozhraní java.lang.Comparable.



Obrázek 4.6: Class diagram vektorů

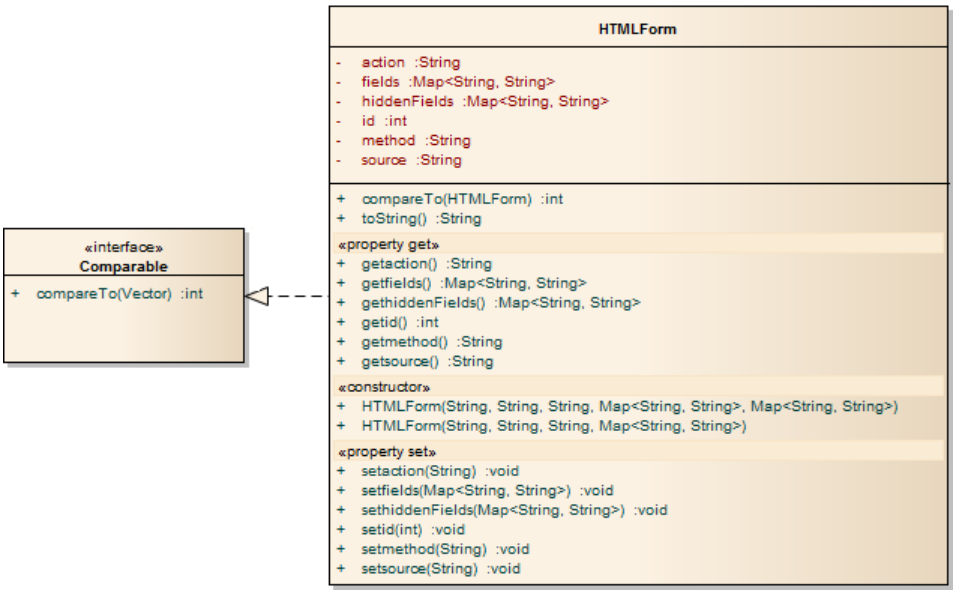
Pro vyhodnocení Cross Site Scripting a HTTP Response Splitting zranitelností je potřeba pouze jedna hodnota vektoru, proto třídy XSSVector a CRLFVector přidávají pouze jeden atribut typu String pro uložení hodnoty.

Pro vyhodnocení SQL Injection je potřeba provést požadavek pozitivní, negativní a neutrální. Z tohoto důvodu přidává třída SQLIVector atribut values, což je jednorozměrné pole řetězců – na indexu 0 je uložen pozitivní vektor a na indexu 1 negativní vektor.

Clickjacking je vyhodnocován analýzou hlaviček – nemá tedy vlastnost value. Instance třídy pak slouží pouze k uložení podrobností o zranitelnosti.

4.6 Reprezentace formuláře

Formuláře mohou být do programu zadány ručně pomocí textových polí, nebo mohou být získány ze vstupního proudu. Reprezentace HTML formuláře pomocí mapy řetězců v prvním případě, nebo pomocí stromu v druhém případě není vhodná pro další zpracování. Z tohoto důvodu byla navržena třída HTMLForm (Obrázek 4.7).



Obrázek 4.7: Class diagram HTML formuláře

Třída má vlastnosti pro uložení všech nutných informací o formuláři pro pozdější odeslání – hodnota atributu action, mapa polí, mapa skrytých polí a metoda. Ve vlastnosti source je uloženo URL, ze kterého daný formulář pochází. Tato informace je nutná pro pozdější aktualizaci CSRF tokenů. Pokud je formulář zadán ručně, tak je hodnota stejná jako ve vlastnosti action.

Aby nebylo nutné na více místech v aplikaci rozdělovat pole formuláře dle typu na skrytá a ostatní, tak má třída HTMLForm dva konstruktory. První konstruktor přebírá výše zmíněné vlastnosti s oddělenými mapami pro skrytá a ostatní pole. Druhý konstruktor přebírá téže parametry, ale jen jednu mapu parametrů. Před vytvořením instance ji rozdělí dle uložené hodnoty do dvou map – pokud hodnota není null, tak je pole považováno za skryté.

Kvůli porovnávání formulářů je implementováno rozhraní Comparable.

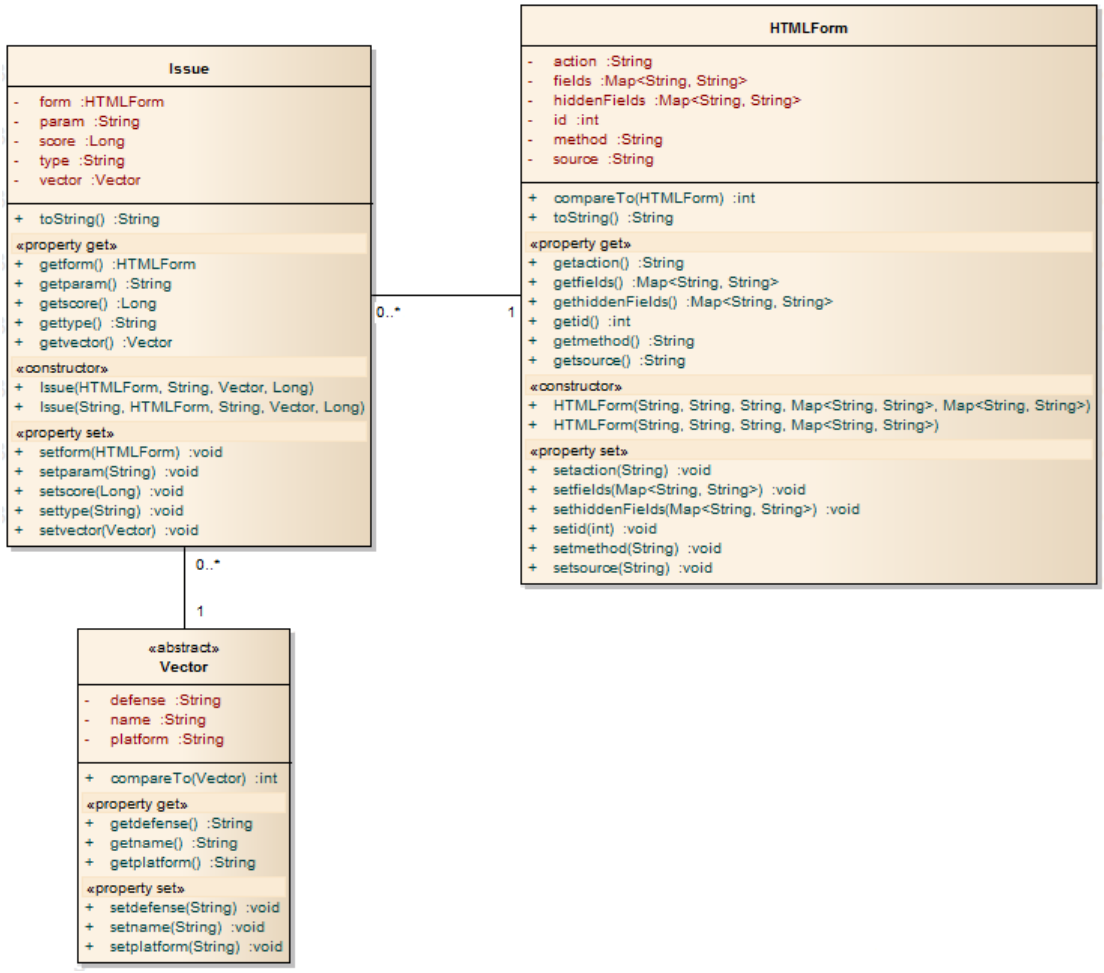
4.7 Reprezentace výsledků

Po otestování jednoho parametru formuláře určitým vektorem je nutné uložit výsledek a později ho uživateli nástroje prezentovat ve vhodném formátu. Aby byl vývojář schopen identifikovat a následně napravit chyby v aplikaci, tak potřebuje následující informace:

- Atribut action formuláře (URL)
- Název parametru
- Typ vektoru

- Hodnotu vektoru
- Popis obrany proti vektoru (pokud je přítomen)

Instance třídy Issue (Obrázek 4.8) je vytvářena pro každou testovanou dvojici vektor a parametr. Spojení s příslušným formulářem a vektorem je řešeno referencí. Pomocí těchto vazeb lze při zobrazení výsledků získat všechny podrobnosti o testovaném formuláři a použitém vektoru. Každá instance třídy Issue má přiřazen právě jeden vektor a formulář.



Obrázek 4.8: Uložení výsledků

Takto jsou uloženy výsledky všech vektorů aplikované na všechny parametry, tedy i negativní výsledky. Ukládat všechny informace může sice být poměrně paměťově náročné, ale zase lze snadno identifikovat parametry, jejíchž výsledky mají být hledány na dalších místech při problému perzistentních útoků.

Výsledky mohou být uživateli prezentovány v tabulce, listu nebo stromu. Při zobrazování dat ve stromu je pomocí knihovny Glazed Lists vytvářena postupně cesta ze zobrazovaných objektů. V každé nižší úrovni je obsaženo více podrobností (je přítomen atribut navíc a ostatní jsou null). Glazed Lists také automaticky transformují index vybrané pozice ve stromu na pozici v listu.

Bylo by možné použít složitější strukturu tříd tak, aby byl snížen počet duplicitních informací, ale při použití tohoto přístupu by bylo mnohem obtížnější v prezentační vrstvě vypisovat

výsledky. Strukturu by bylo nutné transformovat na samostatné objekty (něco na způsob třídy Issue) a z nich následně vytvářet pomocí knihovny Glazed Lists cestu a zobrazit výsledný strom.

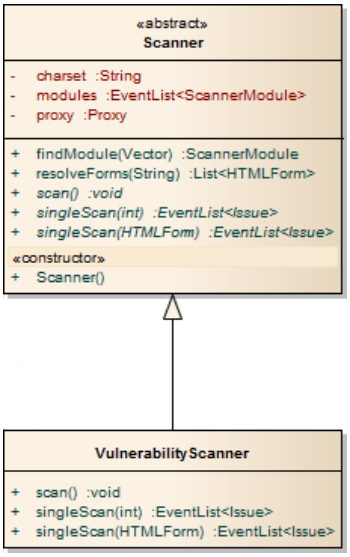
Kvůli výhodám poskytovaným knihovnou Glazed Lists byla zvolena jednodušší varianta.

4.8 Scanner

Abstraktní třída Scanner definuje vlastnosti nutné pro nastavení vykonávání požadavků (znaková sada, adresa proxy serveru) a operace pro testování.

K nastavení adresy proxy serveru slouží vlastnost proxy a k nastavení znakové sady vlastnost charset. Výchozí znakovou sadou je UTF-8.

Poskytuje operace, které jsou používány prezentační vrstvou. Scanner má reference na listy obsahující vektory, formuláře a výsledky. Pomocí metody scan() je proveden test všech formulářů uložených v příslušném listu, pomocí scan(int) je otestován formulář s určitým indexem. Pro otestování konkrétního formuláře slouží metoda scan(HTMLForm). Operace jsou označeny stereotypem abstract a jsou implementovány až v potomcích této třídy.



Obrázek 4.9: Class diagram scanneru

4.9 Architektura aplikace

Nástroj je navržen dle vícevrstvé architektury . Díky tomuto přístupu, který odděluje aplikační logiku od prezentace, je možné snadno přetvořit aplikaci na webovou službu, webovou aplikaci či použít pro grafické uživatelské rozhraní novější technologii.

Prezentační vrstva byla napsána pomocí SWING za použití knihovny Javabuilder. Tato knihovna umožňuje popsat rozhraní aplikace pomocí jazyku YAML[2]. Na této vrstvě je vytvářeno samotné rozhraní, které je prezentováno uživateli aplikace.

Obchodní vrstva tvoří jediný vstupní bod aplikace, prezentační vrstvě poskytuje samotné operace, které je schopna aplikace provádět.

[2]YAML Ain't Markup Language je formát pro serializaci strukturovaných dat

Integrační vrstva umožňuje poskytuje nadřazené vrstvě rozhraní pro práci s daty. Na této vrstvě se nacházejí třídy VectorDAO, FormDAO a IssueDAO. Pomocí VectorDAO jsou spravována data testovacích vektorů, pomocí FormDAO formulářů a pomocí IssueDAO jsou spravovány výsledky testů.

Kapitola 5

Implementace nástroje

V této kapitole jsou popsány nejdůležitější části implementace nástroje – načítání vektorů ze souboru, provádění HTTP požadavků, procházení DOM reprezentace HTML dokumentu, správa cookies a vyhodnocování vektorů. Dále jsou popsány jednotlivé vrstvy architektury aplikace.

5.1 Vektory

5.1.1 Vector

Ve třídě Vector jsou definovány atributy name, platform a defense. Atribut name obsahuje název vektoru uložený jako řetězec. Atribut platform obsahuje jednotlivé zranitelné systémy, které jsou oddělené čárkou. Možný popis obrany proti vektoru je uložen v atributu defense.

Samotné provedení testu zajišťují příslušné moduly odvozené od ScannerModule.

Ve třídě ScannerModule je definována abstraktní metoda testparam(HTMLForm, String, Vector, long, boolean, HttpSender), která slouží k otestování jednoho parametru formuláře . Tuto metodu musejí implementovat všechny odvozené třídy.

K otestování jednoho formuláře slouží metoda testform(HTMLForm, Vector, boolean, HttpSender) vracející mapu String-Long a testForm(HTMLForm, Vector, boolean, HttpSender), která vrací list instancí třídy Issue.

Nejdříve je vytvořena mapa výsledků String-Long. V rámci testování formuláře jsou postupně generovány náhodné značky identifikující test a volána metoda testparam(HTMLForm, String, Vector, long, boolean, HttpSender).

Do mapy jsou postupně ukládány výsledky testů – pokud metoda testparam(HTMLForm, String, Vector, long, boolean, HttpSender) vrací logická pravda, tak je do ní uložena dvojice název parametru a číslo 0. V opačném případě je uložena hodnota značky identifikující příslušný test.

Metody provádějící test formulářů jsou společné pro všechny třídy vektorů.

5.1.2 XSSVector

Třída XSSVector je potomkem třídy Vector. Pro testování na Cross Site Scripting je nutná pouze jedna hodnota vektoru, která je uložena jako řetězec ve vlastnosti value.

Otestování jednoho parametru na zranitelnost XSS je prováděno metodou testparam(HTMLForm, String, String, Vector, long, boolean, HttpSender), ze třídy ReflectedXSSModule

Nejdříve je vytvořena mapa pro provedení testu, poté jsou přidána všechna pole formuláře z instance HTMLForm a všechna jsou nastavena na nějakou hodnotu (například na náhodný řetězec).

Pokud je parametr csrf logická pravda, tak jsou aktualizována skrytá pole formuláře a cookies. Tento postup je prováděn kvůli možné obraně formuláře proti útoku CSRF.

Následně je do do testovaného parametru nastavena hodnota vektoru s případnou značkou a je proveden HTTP požadavek. V odpovědi je vyhledávána právě tato hodnota. Pokud se v získaném dokumentu vyskytuje, tak je parametr považován za zranitelný a je vrácena logická pravda.

5.1.3 CRLFVector

Třída CRLF vektor má oproti předkovi Vector navíc vlastnost value, ve které je uložena hodnota konkrétního vektoru.Ve většině případů bude vypadat podobně jako výpis 5.1.

Pomocí tohoto vektoru lze webovou aplikaci testovat na útoky založené na injekci hlavičky – HTTP Response Splitting, Cross User Defacement či Cache Poisoning.

Samotné provedení testu zajišťuje třída CRLFModule.

Test na CRLF Injection se liší od testu na XSS pouze ve vyhodnocovací části – je předpokládáno, že v odpovědi bude pouze značka a část vektoru. V případě platnosti tohoto předpokladu je parametr zranitelný a je vrácena logická pravda.

Listing 5.1: CRLF Injection

```
1 foo[LF]
2 Content-length: 0[LF]
3 HTTP/1.1 200 OK[LF]
4 Content-Type: text/html[LF]
5 Content-length: 47[LF]
6 [LF]
7 <html>Hacked!</html>
```

5.1.4 SQLIVector

Pro potřeby testování je ke zděděným vlastnostem od předka přidáno jednorozměrné pole řetězců, které obsahuje na první pozici vektor pro pozitivní požadavek a na druhé pozici vektor pro negativní požadavek. Neutrální hodnotu vektoru není nutné ukládat, poněvadž ji lze odvodit z testovaného formuláře.

Otestování jednoho parametru na zranitelnost SQL Injection je prováděno metodou testparam(HTMLForm, String, String, Vector, long, boolean, HttpSender), ze třídy SQLIModule.

Nejdříve je vytvořena mapa pro provedení testu, přidána jsou všechna pole formuláře z instance HTMLForm a všechna jsou nastavena na nějakou hodnotu (například na náhodný řetězec).

Pokud je parametr csrf logická pravda, tak jsou aktualizována skrytá pole formuláře a cookies. Tento postup je prováděn kvůli možné obraně formuláře proti útoku CSRF.

Následně jsou postupně nastaveny odpovídající hodnoty vektoru s případnou značkou marker a provedeny tři požadavky – pozitivní (p), negativní (n) a neutrální (o). Před prováděním každého požadavku je nutné aktualizovat CSRF token a cookies.

Vyhodnocení zranitelnosti spočívá v hledání podobností získaných odpovědí. Jsou spočteny Lehvensteinovy vzdálenosti[1] dvojic odpovědí. Pokud platí

$$lehvenstein(p, o) > lehvenstein(n, o)$$

tak je parametr vyhodnocen jako zranitelný a je vrácena logická pravda.

5.1.5 CLJVector

Pro testování na zranitelnost Clickjacking není nutná testovací hodnota – testování spočívá v analýze hlaviček z HTTP odpovědi.

Otestování jednoho parametru na zranitelnost Clickjacking je prováděno ve třídě CLJModule metodou testparam(HTMLForm, String, String, Vector, long, boolean, HttpSender).

Nejdříve je vytvořena mapa pro provedení testu, přidána jsou všechna pole formuláře z instance HTMLForm a všechna jsou nastavena na nějakou hodnotu (například na náhodný řetězec).

Následně je proveden GET požadavek. Pokud se v hlavičce získané odpovědi vyskytuje hlavička X-FRAME-OPTIONS nastavená na hodnotu DENY, tak je parametr považován za nezranitelný a je vrácena logická nepravda. V případě absence této hlavičky nebo jiné hodnoty je vrácena logická pravda.

5.2 Podpůrné třídy pro vektory

5.2.1 VectorStream

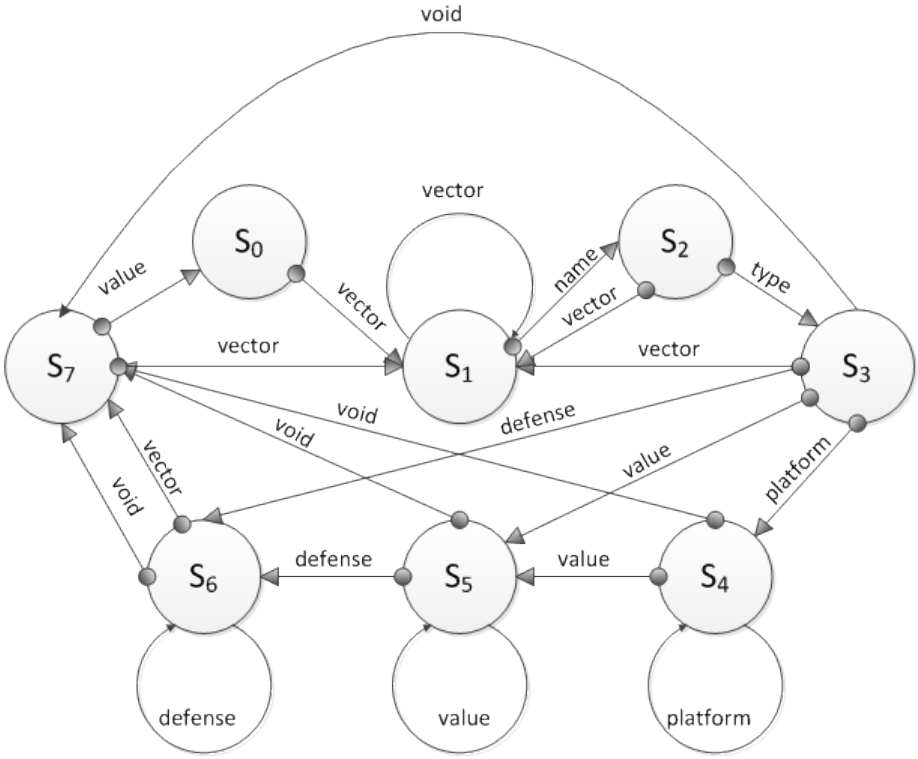
Třída VectorStream slouží k načítání vektorů ze souboru. Pro načtení jednoho vektoru poskytuje veřejnou metodu read, která vrací instanci načteného vektoru. K uzavření proudu slouží metoda close.

Interně VectorStream používá BufferedReader, pomocí kterého načítá řádky ze souboru. Pro parsování vektoru z textového formátu je použit stavový automat (Obrázek 5.1). Stavový automat je tvořen dvourozměrným polem, které představuje matici sousednosti.

V poli jsou uloženy v místě existence hrany regulární výrazy (Tabulka 5.1) a na ostatních pozicích je null. Aktuální stav je uložen v proměnné „state“.

K zakódování stavů by stačilo použít i jednorozměrné pole. Aby však byl dodržen definovaný formát, tak by bylo nutné přidat mnohem složitější logiku pro nalezení dalšího stavu a „hardcoded“ pravidla. Řešení dvourozměrným polem je elegantnější, snadno rozšiřitelné a používá mnohem jednodušší logiku pro přechod do dalšího stavu.

[1] Lehvensteinova vzdálenosti je počet operací, které je nutné provést k převedení řetězce A na řetězec B.



Obrázek 5.1: Stavový automat

Tabulka 5.1: Regulární výrazy

název hrany	regulární výraz
VECTOR	\s*vector1\s*
NAME	\s*name\s?=\s?\S+[\s?\S]*
TYPE	\s*type\s?=\s?\S+
PLATFORM	\s*platform\s?=\s?\S+[\s?\S]*
VALUE	\s*value\s?=\s?(\S || \s)+
DEFENSE	\s*defense\s?=\s?\S+[\s?\S]*
VOID	\s*

Pokud je v souboru uložena nekompletní podoba vektoru, tak automat s následujícím načteným klíčovým slovem „vector“ zahodí nekompletní data a začne načítat následující vektor.

Při vytvoření instance VectorStream se automat nachází ve stavu S_0. Po načtení řádku je zavolána metoda „findNextState“, která z parametru a aktuálního stavu určí následující stav, do kterého má automat přejít. Následuje provedení operací příslušejících každému stavu – pro S_1 vymazání načtených dat a pro S_2 až S_6 uložení hodnot.

Když automat dojde do stavu S_7, tak je vytvořena instance vektoru. Díky polymorfismu lze se všemi druhy vektorů pracovat obdobným způsobem, tedy proto je vracen typ Vector. Instance konkrétního potomka třídy Vector je vytvářena pomocí návrhového vzoru „Factory“ – továrním objektem VectorFactory.

Metoda findNextState(String) slouží k nalezení následujícího stavu. Postupně je procházen i-tý řádek matice, kde i je aktuální stav. Pokud regulární výraz na indexu (i,j) odpovídá načtenému řetězci, tak je okamžitě vrácena hodnota j – bere se tedy první shoda. Pokud následující stav není nalezen, tak je vrácena -1. Zotavení automatu z nedefinovaného stavu je provedeno uvedením

do stavu S_0.

5.2.2 VectorFactory

Po načtení veškerých potřebných dat VectorStreamem vyvstává problém, jak rozhodnout, kterou třídu konkrétního vektoru použít k vytvoření nové instance. Jelikož všechny vektory dědí od třídy Vector, tak lze tento problém řešit pomocí návrhového vzoru továrna.

VectorStream má uloženou referenci na tovární objekt. Pokud má načtena veškerá potřebná data ze souboru, neboli se nachází v konečném stavu S_7, tak zavolá tovární metodu createVector(String, String, String, String[], String) (Kód 5.2), které předá všechny načtené informace. Interní logika metody rozhodne na základě parametru type o konkrétní třídě, jejíž instance má být vytvořena a vrátí referenci na vytvořenou instanci.

K vytváření vektorů slouží metoda createVector(String, String, String, String[]). V těle metody je dle načteného typu, pomocí klauzule switch, určena třída, od které má být vytvořena nová instance. V případě načtení poškozených nebo nekompletních dat je vyvolána výjimka VectorException.

Listing 5.2: Tovární metoda createVector

```
1  public Vector createVector(String name, String type, String platform,
2              String values[], String defense) throws VectorException {
3          switch(type) {
4              case XSS:
5                  return new XSSVector(name, platform, values[0], defense);
6              case SQLI:
7                  return new SQLIVector(name, platform, values, defense);
8              case CRLF:
9                  return new CRLFVector(name, platform, values[0], defense);
10             default: throw new VectorException(VectorException.ERROR03);
11         }
12     }
```

5.2.3 VectorException

Je potomkem třídy java.lang.Exception a slouží jako univerzální výjimka pro všechny části aplikace, které se týkají načítání a vytváření vektorů.

Dále obsahuje řetězcové konstanty všech chyb, které mohou nastat (Tabulka 5.2). V samotné výjimce jsou uloženy pouze univerzální anglické zprávy. Lokalizace může být obstarána až na prezentační vrstvě.

5.3 Obsluha HTTP a HTTPS

5.3.1 HttpSender

Třída HttpSender je jednou z hlavních částí aplikace. Vytváří vyšší úroveň abstrakce nad protokoly HTTP a HTTPS[2] – umožňuje provést požadavek voláním jediné metody, automaticky kóduje parametry do URL kódování a vybírá protokol dle zadaného URI. Úzce spolupracuje s CookieManagerem, který poskytuje podporu pro cookies.

HttpSender umožňuje použít následující operace:

- Provedení GET požadavku
- Provedení POST požadavku
- Provedení GET požadavku s navrácením proudu
- Připojení přes proxy server
- Použití šifrovaného protokolu HTTPS

Dále je možné nastavit výchozí znakovou sadu, kešování obsahu a časový limit požadavku.

V případě, že je požadavek prováděn metodou GET, tak je nutné zakódovat parametry s jejich hodnotami a následně je přidat do URI. Vstupní URI může být ve formátu bez parametrů – je pak přidán „?“ a zakódované parametry, nebo může být ve formátu s parametry a pak je potřeba přidat symbol „&“. Identifikace typu je prováděna pomocí regulárních výrazů.

Parametry jsou předávány v mapě Map<String,String>. Postupně je konstruována posloupnost (Kód 5.3) zakódovaných dvojic „název=hodnota“ a symbolů „&“. Ke kódování do URL je použita třída URLEncoder z balíčku java.net. V případě, že hodnota parametru je prázdný řetězec, tak je nutné jej též zakódovat. Pokud je však jeho hodnota null, tak přidán do posloupnosti není.

Listing 5.3: Parametry zakódované do URL

```
1 http://localhost/foo/bar.php?message=foo&author=bar&filtered=f%F9b%E1r&
      action=show
```

Privátní metoda makeStream(String, Map<String,String>, Map<String,String>, String) poskytuje neobecnější funkcionalitu. Všechny ostatní metody používají výstupní proud vracený touto metodou.

[2]Hypertext Transfer Protocol Secure

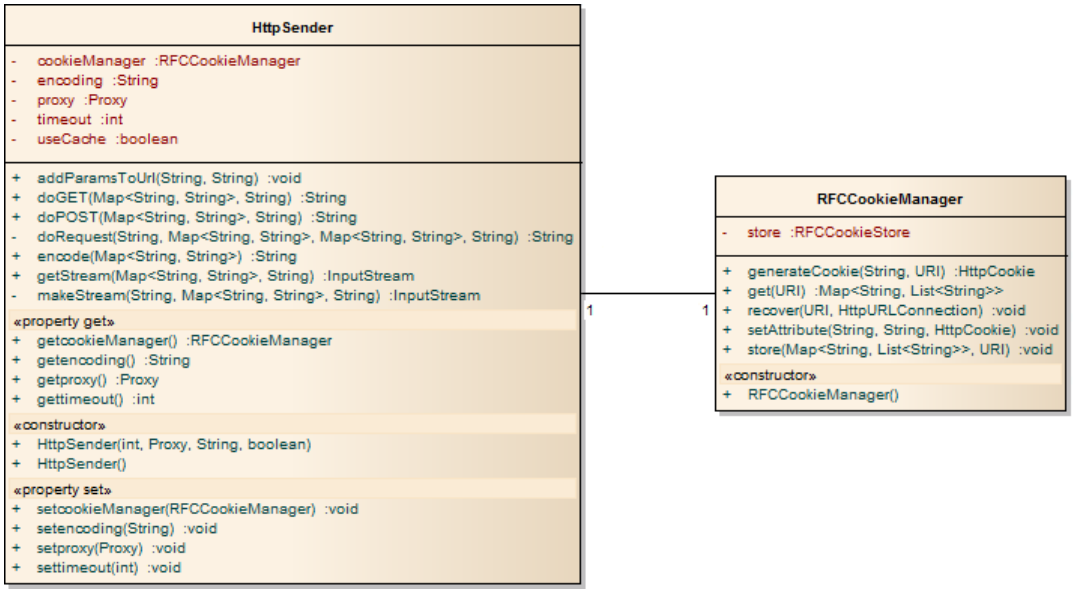
Tabulka 5.2: Chybové zprávy vektoru

identifikátor	zpráva
ERROR01	Value cannot be null or void string.
ERROR02	Number of values for SQLIVector must be be at least two.
ERROR03	Uknown type of vector.
ERROR04	Vector file is corrupted.

Nejdříve jsou zakódovány parametry do textového URL a následně je vytvořen objekt typu URI. Tento objekt je použit pro vytvoření přímého HTTP spojení, nebo spojení přes proxy server. Podle přítomnosti „http“ nebo „https“ v URL je pomocí regulárních výrazů určeno, zda bude vytvořeno šifrované, nebo nešifrované spojení. Před samotným otevřením spojení je nejdříve nutné nastavit hlavičky. Po otevření spojení je dle kódu odpovědi provedena akce – pro kód 200 jsou aktualizovány cookies a vrácen proud. V ostatních případech je vygenerována výjimka.

HttpSender umožňuje provést požadavky za použití metod GET i POST. Nejdříve je získán proud pomocí metody makeStream(String, Map<String,String>, Map<String,String>, String) a následně jsou z proudu čteny řádky, které jsou ukládány do bufferu. Obsah bufferu je následně vrácen.

Instance HttpSenderu má uloženu referenci na právě jednu instanci CookieManageru (Obrázek 5.2).



Obrázek 5.2: Třída HttpSender

5.3.2 CookieManager

Z důvodu bezestavovosti protokolů HTTP/HTTPS musejí webové aplikace „ukládat“ stav relace pomocí dalších technologií, jako jsou relačních proměnné či cookies. V případě použití relačních proměnných musí klient odesílat v požadavku na server identifikátor Session ID[3]. Hodnota Session ID může být odesílána v parametru, ale častěji bývá použita cookie. Z tohoto důvodu je nutné, aby nástroj měl implementovánu podporu pro cookies. Bez ní by totiž nebyl schopen otestovat součásti aplikace vyžadující autentizaci.

Dalším důvodem proč je tato podpora nutná, je obrana webových aplikací proti CSRF útokům, která je nejčastěji realizována nějakou formou podepsaných formulářů tokenem. Referenční hodnota může být uložena v relační proměnné – je pak nutné odesílat token v cookie či parametru, nebo jsou srovnávány hodnoty tokenu z cookie a z parametru. Aby tedy mohly být testovány

[3]Informace podle které server identifikuje příslušné relační proměnné patřící klientovi (případně kontext).

podepsané formuláře, tak je podpora cookies nutná téměř pro všechny webové aplikace[4].

Samotná cookie je v programu reprezentována třídou java.net.HttpCookie. Tato třída poskytuje metody pro získání a nastavení jednotlivých atributů cookie, zajišťuje výpočty expirace cookie dle [8] a vytváří textovou podobu cookie vhodnou pro odeslání na server v hlavičkách.

Hlavička odpovědi může obsahovat řetězec „Set-Cookie2:“, za nímž následuje seznam cookies oddělených čárkou. Každá cookie začíná dvojicí název=hodnota a pak mohou následovat jednotlivé atributy s příslušnými hodnotami oddělené středníky. Podle specifikace [11] musí v cookie přijít dvojice název=hodnota přijít jako první. Pokud jsou obsaženy atributy, tak mohou být v jakémkoliv pořadí. Když se stejný atribut vyskytuje vícekrát, tak je brán v potaz pouze první výskyt. Podobným způsobem jsou zpracovávány také hlavičky „Set-Cookie:“.

Listing 5.4: Hlavička obsahující cookies

```
1 HTTP/1.1 200 OK
2 Date: Wed, 21 Mar 2012 13:21:04 GMT
3 Server: Apache/2.2.21 (Win32) PHP/5.3.9
4 X-Powered-By: PHP/5.3.9
5 Set-Cookie: PHPSESSID=j66bojhb8oadn9e6mbnn69hl42; path=/
6 Expires: Thu, 19 Nov 1981 08:52:00 GMT
7 Cache-Control: no-store, no-cache, must-revalidate, post-check=0,pre-check=0
8 Pragma: no-cache
9 Set-Cookie: foo=bar; expires=Wed, 12-Dec-2012 12:12:12 GMT
10 Content-length: 1123
11 Content-Type: text/html
```

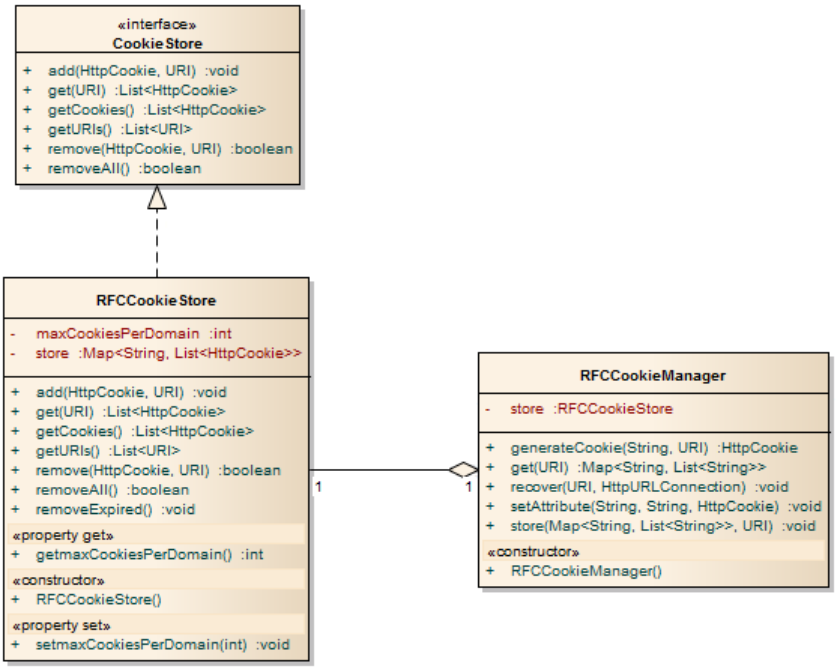
Kladná hodnota uložená ve vlastnosti maxAge udává počet sekund, po kterých je cookie platná. Pokud je hodnota rovna nule, tak by cookie měla být odstraněna. Pokud má být platnost cookie omezena délkou uživatelské relace tj. cookie má vypršet se zavřením klientské aplikace, pak je nutné nastavit maxAge na zápornou hodnotu. Toto chování je definováno ve třídě java.net.HttpCookie .

Podle normy RFC2965 [11] má přednost direktiva „max-age“ před direktivou „expires“. Pokud je tedy nalezen atribut „expires“ a vlastnost maxAge je již nastavena, tak není třeba hodnotu měnit. Pokud je nalezena „max-age“, tak je nutné změnit expiraci vždy. V případě, že není přijat atribut max-age, tak je implicitně nastavena platnost cookie po celou dobu relace. Čas expirace je vypočítán na základě pravidel pro výpočet stáří a expirace definovaných v rfc2616. V případě přítomnosti direktivy „max-age“ je přímo nastavena platnost v sekundách (dle přijaté hodnoty) a v případě „expire“ je vypočtena z přijatého data převedením na sekundy.

Ověření, zda cookie již expirovala, zajišťuje třída HttpCookie, která poskytuje metodu isExpired vracející true, pokud cookie expirovala, jinak false.

K samotnému uložení cookies slouží třída RFCCookieStore.

[4]Výjimkou by byla aplikace předávající Session ID v parametru, která má referenční token v relační proměnné a odesílá token v parametru.



Obrázek 5.3: Správa cookies

Přidání cookie

1. Získání cookies z hlaviček
2. Nastavení implicitních hodnot atributů
3. Vyfiltrování duplicitních atributů
4. Nastavení atributů
5. Přidání do uložiště

Získání cookie

1. Získání cookies z uložiště (řeší RFCCookieStore)
2. Přidat cookies do listů dle verze (Nestcape nebo RFC2965)
3. Nastavení hlaviček dle listů

Všechny části třídyCookieManager se snaží co nejvíce naplňovat požadavky na klienta norem RFC2109 [10], RFC2965 [11] a RFC2965 [11]. Pro bezpečnostní skener však není nutné implementovat určité požadavky – například bezpečnostní pravidla (kontrola portů, kontrola domény,...).

5.3.3 RFCCookieStore

Třída RFCCookieStore implementuje rozhraní java.net.RFCCookieStore. Zajišťuje uložení cookies do příslušných uložišť domény, poskytuje metody pro práci s cookies - přidání, smazání a získání cookie jedné či všech cookie příslušejících serveru.

Poněvadž specifikace [11] určuje chování v případě expirace cookie, tak je navíc implementována metoda k odstranění expirovaných cookies. Tato operace má být prováděna vždy před odesláním cookies. Navíc jsou také expirované cookies odstraňovány i před přidáním cookie – z důvodu omezení počtu cookies na doménu.

Maximální počet cookies na doménu je podle [11] 20. RFCCookieStore má tuto hodnotu nastavenu implicitně na 50, což by měla být dostačující hodnota. Pokud by tato hodnota nedostačovala, lze ji změnit nastavením vlastnosti maxCookiesPerDomain.

Přidání cookie do uložiště

1. Získání uložiště cookies
   Z přidávané cookie je získána hodnota atributu „domain“. Pokud pro hodnotu atributu neexistuje uložiště, tedy nebyla ještě uložena žádná cookie z daného serveru, tak je vytvořena nová instance listu sloužící jako uložiště. Následně je v obou případech vrácena reference na příslišný list.

   Specifikace [11] říká, že porovnávání domén má být case insensitive. Z tohoto důvodu je doména (resp. host část domény) transformována do malých písmen.

2. Přidání a aktualizace cookie
   U všech cookie v uložišti domény je porovnána vlastnost „name“ a následně vlastnost „path“. Pokud jsou stejné obě hodnoty, tak se cookie v uložišti již nachází a je nutné ji aktulizovat – smazat a přidat novou hodnotu. V případě, že se shoduje pouze název cookie, tj. rozsah platnosti (scope) daný cestou je odlišný, tak je cookie pouze přidána.

   Nová cookie, která se v uložišti zatím nenachází, je taktéž přidána.

   Všechny atributy byly již dříve nastaveny v třídě CookieManager. Třída RFCCookieStore zajišťuje pouze přidávání, odebírání a získávání platných cookies pro daný rozsah.

Získání cookie z uložiště

1. Získání uložiště cookies
   Název hostitele z URI (s přidanou vedoucí tečkou) je nutné porovnávat vůči doméně cookie case insensitive.

2. Ověření cesty
   Je povoleno vracet pouze cookies, které mají cestu méně konkrétní než je cesta uvedená v URI požadavku. Například je-li URI požadavku http://www.foobar.org/foo/bar/, tak je možné přidat cookie s cestou /, /foo/ a /foo/bar/, nikoliv /bar/ či /foo/bar/foobar/.

3. Ověření expirace
   Stačí volat metodu isExpired(). Vyhodnocení a vše ostatní zajišťuje třída HttpCookie.

4. Ověření portů
   Cookie lze přidat pouze pokud portu z URI odpovídá alespoň jeden port ze seznamu uvedeném v cookie.

5. Ověření domény

6. Vrácení cookies

5.3.4 TargetSeeker

Třída TargetSeeker slouží k vyhledávání formulářů na zadaném URL. K vyhledávání formulářů používá třídu JTidy z knihovny Apache Commons. JTidy je HTML/XHTML[5] parser, který v konstuktoru přebírá instanci na vstupní HTTP proud. Z tohoto proudu následně čte data a vytváří stromovou reprezentaci hypertextového dokumentu. Pro samotné procházení dokumentu je možné použít standardní grafové algoritmy.

TargetSeeker poskytuje veřejnou metodu seek(String, HttpSender) přebírající jako parametry URL stránky, na které má být formulář vyhledáván a referenci na instanci HttpSenderu. V metodě jsou provedeny následující operace:

- Vytvoření instance JTidy
- Získání vstupního proudu z HttpSenderu
- Vytvoření mapy parametrů
- Hledání formulářů v dokumentu
- Vytvoření listu s nalezenými formuláři

K hledání formulářů je použit algoritmus průchodu grafu do hloubky. Nejdříve je zavolána privátní metoda visit(Node, int, Map<String, Map<String, String>>) na kořenový element získaný pomocí JTidy s hloubkou 0. Při navštívení vrcholu jsou vyhledávány uzly s názvem „form“, „input“ a „a“. Pro každý nalezený formulář je do předem vytvořené mapy formulářů přidána ještě mapa parametrů. Dále je přidán speciální parametr s klíčem null a hodnotou parametru action. Tato informace je nutná pro následovné vytvoření objektů HTMLForm.

V případě nalezení uzlu s názvem „input“ jsou prohledávány uzly zpět směrem ke kořenovému elementu. Takto je identifikován nadřazený uzel „form“. Díky zpětnému prohledávání nevadí, pokud uzel „form“ není přímým rodičem uzlu „input“. Kvůli zpětnému vyhledávání nebude provedeno zbytečně moc operací za předpokladu, že příslušné uzly nejsou příliš vzdáleny. Pokud uzel „input“ nebude mít jako přímého předka „form“, tak bude vyhledáváno až ke kořenu – bude provedeno $n - 1$ operací při zpětném prohledávání, kde n je hloubka větve.

Po úspěšném určení formuláře je dle jeho atributu „action“ získána mapa parametrů a následně jsou parametry přidány do mapy. Parametr, který nemá v této chvíli známou hodnotu je zde uložen jako dvojice název - null.

Pokud je nalezen uzel s názvem „a“, tak je přidán do listu odkazů. Po dokončení procházení jsou sesbírané odkazy normalizovány převedením z relativních na absolutní URL. Ta jsou určena z nalezeného relativního a zdrojového URL.

Provedení těchto operací je zavolána metoda visit(Node, int, Map<String, Map<String, String>>), s hloubkou větší o jedna, postupně na všechny potomky.

[5]Extensible Hypertext Markup Language

Dále poskytuje metodu actualizeForm(HTMLForm, HttpSender), která slouží k aktualizaci skrytých polí formuláře. Též používá k průchodu stromu dokumentu metodu visit(Node, int, Map<String, Map<String, String>>), ale vrací pouze vstupní formulář s aktualizovanými hodnotami. Při této operaci zároveň dojde k nastavení cookies.

5.4 Integrační vrstva

5.4.1 VulnerabilityScanner

Jedná se o třídu, která provádí resp. koordinuje samotné testování formulářů vektory. Třída VulnerabilityScanner je potomkem třídy Scanner (Obrázek 4.9) a implementuje metody pro otestování jednoho formuláře:

- singleScan(int) – otestuje formulář na daném indexu v objektu FormDAO
- singleScan(HTMLForm) – otestuje předaný formulář
- singleScan(String, Map<String, String>, Map<String, String>, String) – vytvoří novou instanci třídy HTMLForm a zavolá singleScan(HTMLForm)

Pro otestování všech formulářů uložených v objektu FormDAO slouží metoda scan().

Při testování formuláře je nejdříve vytvořen list výsledků, postupně je formulář otestován všemi vektory a výsledky jsou zapisovány do listu, který je nakonec předán zpět v návratové hodnotě.

Při každém volání metody je vytvořena nová instance HttpSenderu. V budoucnu by bylo vhodné zavést recyklování použitých instancí HttpSenderu. Každá instance by měla stav používaný a nepoužívaný. Při dokončení požadavku by byla instance vrácena do poolu, ze kterého by mohla být vyzvednuta jiným vláknem.

Testy jsou aplikovány pouze na pole formuláře, která nejsou označena jako skrytá, to jest na pole nacházející se v mapě fields v instanci třídy HTMLForm. Skryté parametry umístěné v mapě hiddenFields a parametry, které jsou součástí URL testovány nejsou. V případě, že by bylo nutné skrytá pole otestovat, tak by nebyl problém je přidat do mapy testovaných polí.

5.4.2 Kolekce pro uložení dat v DAO

Všechny objekty pro přístup k datům (DAO[6]) používají pro uložení jednotlivých hodnot BasicEventList z knihovny Glazed Lists. Aby mohly být změny modelů automaticky propagovány do pohledů, tak jsou tyto listy pozorovatelné pomocí ObservableElementList. Ten využívá možností třídy PropertyChangeSupport a z tohoto důvodu musí DAO a příslušné třídy v kolekci v deklaraci kód 5.5.

Aby bylo možné sdílet listy mezi více vlákny, tak bylo nutné zajistit vláknovou bezpečnost. Toho bylo docíleno pomocí dekorovaného listu ThreadSafeList, kterému je předána reference na původní BasicEventList, ThreadSafeList následně automaticky získává a uvolňuje zámky zdrojů.

[6]Data Access Object

Při použití tohoto přístupu není možné provádět složitější operace atomicky. Další nevýhodou je nižší výkon než při manuálním použití zámků [17]. U tohoto nástroje však ani jedna z nevýhod nijak neomezuje jeho použití.

Listing 5.5: Fragment kódu pro použití PropertyChangeSupport

```
1 public void addPropertyChangeListener(PropertyChangeListener l) {
2   support.addPropertyChangeListener(l);
3 }
4
5 public void removePropertyChangeListener(PropertyChangeListener l) {
6   support.removePropertyChangeListener(l);
7 }
8
9 private final PropertyChangeSupport support =new PropertyChangeSupport(this)
      ;
```

5.4.3 FileVectorDAO

Implementuje operace z rozhraní VectorDAOInterface. Třída FileVectorDAO slouží ke správě vektorů, které byly načteny ze souboru. K načtení vektorů ze souboru v metodě load(String) používá třídu VectorStream. Instance třídy Vector jsou uloženy v kolekci EventList.

5.4.4 FormDAO

Slouží k uložení a správě formulářů reprezentovaných objekty HTMLForm. Objekty formulářů jsou uloženy v kolekci EventList.

5.4.5 IssueDAO

Třída IssueDAO ukládá výsledky reprezentované objekty Issue do listu EventList. Kolekce dat z tohoto DAO je zobrazována v prezentační vrstvě. Implementuje rozhraní IssueDAOInterface.

5.5 Obchodní vrstva

Ve vrstvě obchodní logiky se nachází jediná třída – WVSFacade. Tato třída zjednodušuje strukturu logiky integrační vrstvy při pohledu shora, poskytuje operace nadřazené prezentační vrstvě a stará se o vytvoření objektů Scanner, IssueDAO, FormDAO a VectorDAO.

5.6 Prezentační vrstva

5.6.1 Globální proměnné

Třída Globals slouží k přístupu ke globálním proměnným, které mají být přístupné ve všech třídách prezentační vrstvy. Aby nemohlo být vytvořeno více instancí, tak byla třída navržena podle návrhového vzoru Singleton. Instanci lze získat zavoláním statické metody getInstance().

Obrázek 5.4: Class diagram business facade

V konstruktoru této třídy je vytvořena nová instance třídy WVSFacade z obchodní vrstvy. Tím jsou následně vytvořeny i objekty nižších vrstev.

5.6.2 Hlavní okno

Je tvořeno záložkami pro poloautomatický test, ve kterém lze provést test získaného formuláře nebo formuláře z textových polí, záložkou pro automatický test a záložkou pro zobrazení všech výsledků a podrobností k nim.

Ostatní operace jako je načítání vektorů či spouštění testu lze provádět z roletového menu.

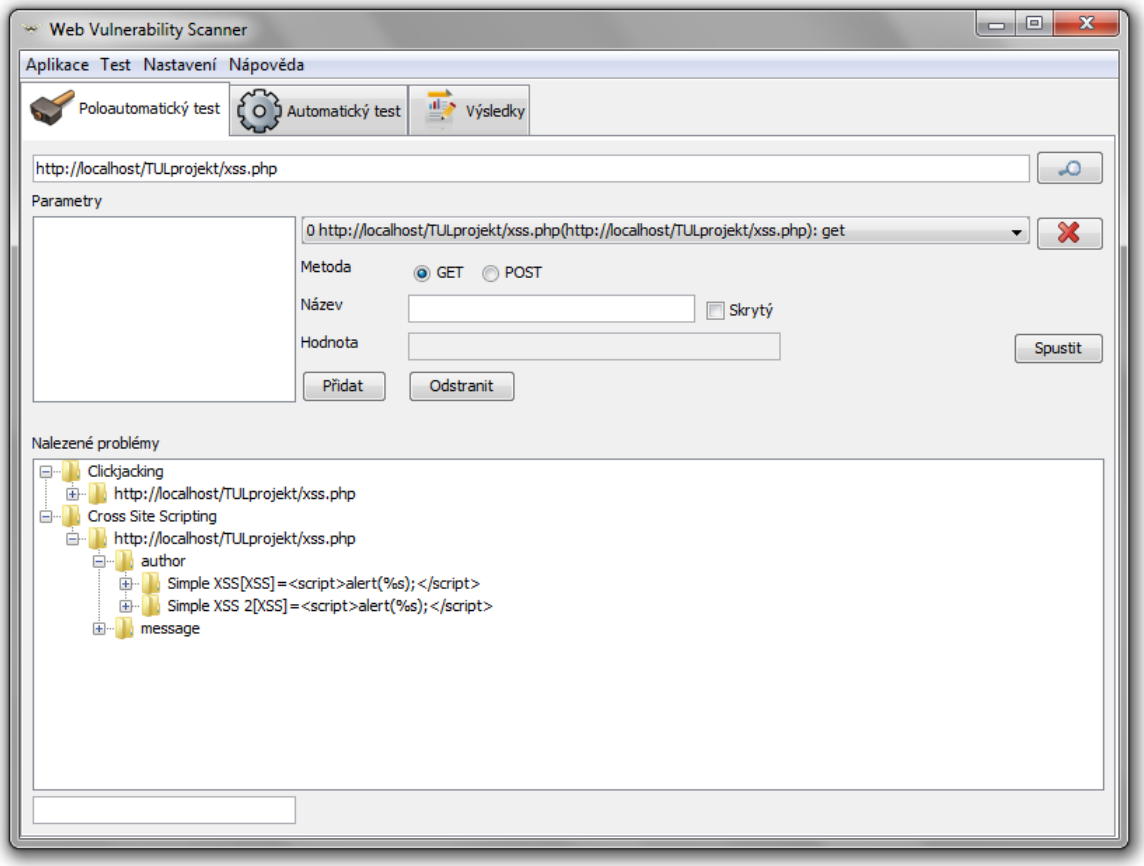
Popis grafického uživatelského rozhraní je zapsán pomocí jazyku YAML v souboru MainWindow.yml, který je s třídou MainWindow spojen pomocí anotace @BuildFile("MainWindow.yml").

Výsledky jsou vypisovány do dvou stromů. Pro samostatný formulář do stromu v záložce „Poloautomatický test“ a pro všechny testy ve stromu v záložce „Výsledky“. Data pro stromy jsou získávána z integrační vrstvy pomocí instance třídy Globals. Stromy umožňují filtrování výsledků (pomocí třídy FilterList z knihovny GlazedLists) podle typu zranitelnosti, URL formuláře (atribut action) a názvu parametru. Hodnota pro filtrování je získávána z textového pole. Podrobnosti o výsledku testu jsou zobrazeny po označení konkrétního záznamu v panelu napravo od stromu (v záložce „Výsledky“).

Veškeré změny modelu jsou automaticky promítány pomocí BetterBeansBinding a Glazed Lists do pohledu a naopak.

Operace, u kterých je předpokládán delší čas vykonávání (testování, získávání formuláře, načítání vektorů), jsou předávány ke zpracování SwingWorkeru přidáním anotace @DoInBackground(cancelable=true, indeterminateProgress = true, blocking = true) před metodu.

K získání formuláře slouží metoda resolve(BackgroundEvent), která na URL zadaném po-



Obrázek 5.5: Hlavní okno nástroje

mocí textového pole vyhledá formuláře a přidá je do ComboBoxu. Následně může být formulář vybrán a otestován zavoláním metody startScan(BackgroundEvent). Pokud není vybrán žádný formulář, tak se nástroj pokusí vytvořit nový formulář z dat, která byla zadána pomocí textových polí. Výsledek testu je automaticky přidán do stromu výsledků.

Pro načtená vektorů slouží metoda loadVectors(BackgroundEvent). Cesta k souboru s vektory je vybrána pomocí dialogu.

5.6.3 Okno nastavení

Popis grafického uživatelského rozhraní je zapsán v souboru SettingsWindow.yml a logika je ve třídě SettingsWindow.java.

V tomto okně je možné nastavit cookies (normální cookie a SID[7] cookie) a parametry připojení (adresa a port proxy serveru, šifrované spojení určuje URL).

5.7 Zhodnocení

Pomocí tohoto nástroje byl proveden test webové aplikace DVWA. Výsledky testů pro jednotlivá nastavení bezpečnosti jsou uvedeny v tabulce 5.3.

[7]Session ID

Tabulka 5.3: Výsledky testu aplikace DVWA

cíl	úroveň	XSS	SQLI	Clickjacking	poznámky
xss_r	low	1	-	ano	-
xss_r	medium	1	-	ano	-
xss_r	high	0	-	ano	Vyžaduje složitější útoky.
xss_s	low	2	1	ano	Falešně detekované SQLI
xss_s	medium	1	1	ano	Falešně detekované SQLI
xss_s	high	0	1	ano	Falešně detekované SQLI
sqli	low	-	1	ano	-
sqli	medium	1	0	ano	Vyžaduje složitější útoky.
sqli	high	-	0	ano	Vyžaduje složitější útoky.

V případě testu na http://localhost/dvwa/vulnerabilities/xss_s/ byl přidán k vektorům bypass „¨“. Při použití bypassu byl správně označen i parametr txtMessage jako zranitelný.

Při testování zranitelnosti http://localhost/dvwa/vulnerabilities/sqli/ na SQL Injection byly nalezeny při nastavení střední bezpečnosti i chyby XSS. To bylo zapříčiněno přítomností ladících zpráv při tomto nastavení.

Při testování skriptu http://localhost/dvwa/vulnerabilities/xss_s/ byly nalezeny falešně pozitivní nálezy na zranitelnost SQL Injection. Tento skript umožňuje přidávat záznamy do databáze a následně je zobrazit. V tomto případě dojde při provádění testu na SQL Injection k situaci, kdy současný výstup obsahuje testovací hodnoty pro všechny předchozí části vyhodnocování SQL Injection (více v části 5.1.4).

Při nezranitelném vstupu v kombinaci s vypisováním vstupních hodnot je v pozitivní odpovědi obsažena pozitivní a negativní hodnota. Ve výstupu na neutrální požadavek je obsažena pozitivní hodnota, negativní hodnota a neutrální hodnota. Vstup je nesprávně vyhodnocen jako zranitelný, poněvadž si je více podobná neutrální s pozitivní odpovědí.

Pro vyhodnocování SQL Injection by bylo vhodné navrhnout lepší funkci, která by eliminovala závislost výstupu mezi jednotlivými částmi SQL Injection testu. Jednodušší variantou by bylo nejdříve vykonat neutrální požadavek, následně negativní a nakonec pozitivní požadavek.

Při nastavení nejvyšší úrovně bezpečnosti testované aplikace je použito nahrazování řídících znaků příslušnými entitami či jiné filtrování (záleži na modulu). Pro jejich otestování by bylo nutné provést sofistikovanější útoky.

Dále se pomocí tohoto nástroje povedlo nalézt XSS zranitelnost na stránkách www.tul.cz, kde byl zranitelný parametr „vyraz“ (Obrázek B.1). Tento nález byl nahlášen správci univerzitních stránek.

Kapitola 6

Závěr

Nástroj vytvořený v této práci je možné používat při testování jednotlivých modulů webové aplikace. Nástroj umožňuje načíst vektory z textového formátu a následně jimi otestovat manuálně zadané formuláře, či formuláře automaticky získané ze zadaného URL. Testovat lze také formuláře chráněné proti Cross Site Request Forgery pomocí různých derivátů metody podepsání pomocí tokenů.

Výsledky testování jsou uživateli prezentovány v přehledné stromové struktuře, ve které si může zobrazit podrobnosti o provedeném testu a výsledky filtrovat podle zadaných klíčových slov. Nástroj umožňuje z výsledků vytvořit výstupní zprávu formátovanou pomocí HTML.

V současné době je možné bezproblémové vyhodnocování zranitelnosti na Clickjacking, neperzistentní XSS a CRLF Injection. Úspěšnost je dána množinou testovacích vektorů a přítomností bypassů. V případě vyhodnocování SQL Injection mohou být přítomny falešně pozitivní nálezy. Tento jev by mohl být v budoucnu odstraněn změnou vyhodnocovací funkce nebo změnou pořadí vykonávání požadavků, které jsou uvedeny části 5.7.

Nejdůležitější funkcí, která by měla být v budoucnosti přidána, je podpora pro získávání formulářů z přepisovaných URL (například pomocí mod_rewrite).

V současné implementaci schází možnost testovat webovou aplikaci ve více vláknech. Jejím zavedením by se jistě podstatně zvýšila rychlost skenování.

Při dalším pokračování ve vývoji by mohla být přidána podpora pro další typy útoků (např. SQL Injection specifické pro určité databázové servery) a rozšířena množina testovacích vektorů dle aktuálních poznatků z bezpečnosti webových aplikací.

Literatura

[1] BARTH, A. HTTP State Management Mechanism. RFC 6265 (Proposed Standard), April 2011a. Dostupné z: `http://www.ietf.org/rfc/rfc6265.txt`.

[2] BARTH, A. The Web Origin Concept. RFC 6454 (Proposed Standard), December 2011b. Dostupné z: `http://www.ietf.org/rfc/rfc6454.txt`.

[3] BAU, J. et al. State of the Art: Automated Black-Box Web Application Vulnerability Testing. May 2010.

[4] .CCUMINN. Cross Site Request Forgery. *Hakin9*. 2 2008a, 21, s. 58–67. ISSN 1214-7710.

[5] .CCUMINN. Pokročilé techniky XSS. *Hakin9*. 3 2008b, 22, s. 14–24. ISSN 1214-7710.

[6] FANELLI, A. SQL Injection in Action. *Hakin9*. 2008, 3, 6, s. 32–38. ISSN 1733-7186.

[7] FANELLI, A. Keylogger 2.0. *Hakin9*. 2009, 4, 1, s. 22–26. ISSN 1733-7186.

[8] FIELDING, R. et al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Dostupné z: `http://www.ietf.org/rfc/rfc2616.txt`. Updated by RFCs 2817, 5785, 6266.

[9] GAUTAM. The Most Dangerous Attack Of Them All. *Hakin9*. 2012, 7, 1, s. 36–41. ISSN 1733-7186.

[10] KRISTOL, D. – MONTULLI, L. HTTP State Management Mechanism. RFC 2109 (Historic), February 1997. Dostupné z: `http://www.ietf.org/rfc/rfc2109.txt`. Obsoleted by RFC 2965.

[11] KRISTOL, D. – MONTULLI, L. HTTP State Management Mechanism. RFC 2965 (Historic), October 2000. Dostupné z: `http://www.ietf.org/rfc/rfc2965.txt`. Obsoleted by RFC 6265.

[12] LISCI, M. The Real World Clickjacking. *Hakin9*. 2009, 4, 2, s. 42–47. ISSN 1733-7186.

[13] ORTEGA, J. G. Practical Client Side Attacks. *Hakin9*. 2012, 7, 1, s. 10–15. ISSN 1733-7186.

[14] OWASP Foundation. OWASP Top Ten - 2010: The Ten Most Critical Web Application Security Risks. [online], [citováno 15.5.2012]. Dostupné z: `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`.

[15] RYDSTEDT, G. et al. Busting Frame Busting a Study of Clickjacking Vulnerabilities on Popular Sites. In *Web 2.0 Security and Privacy 2010(W2SP)*, May 2010.

[16] WhiteHat Security. Measuring Website Security: Windows of Exposure. [online], [citováno 20.4.2012]. Dostupné z: `https://www.whitehatsec.com/`.

[17] WILSON, J. Glazed Lists Tutorial. [online], [citováno 15.5.2012]. Dostupné z: `http://www.glazedlists.com/documentation/tutorial`.

Příloha A

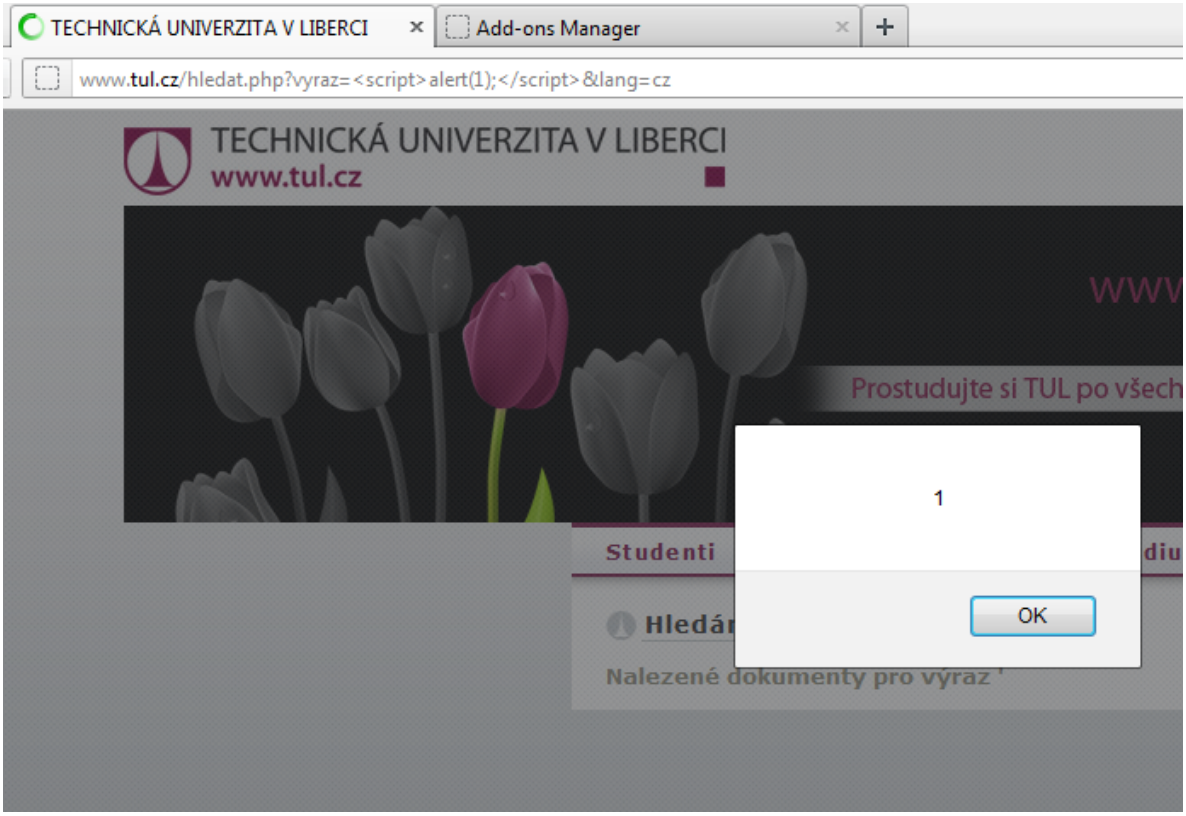
Obsah přiloženého CD

Na přiloženém CD je uložen text práce ve formátu PDF. Dále jsou v adresářích obsaženy:

- dist – přeložená aktuální verze programu
- src – zdrojové kódy
- knihovny – použité knihovny použité s příslušnými licencemi
- testované nástroje – instalační balíčky testovaných aplikací
- testovaná aplikace – v adresáři „dvwa“ je umístěna aplikace Damn Vulnerable Web Application a ve složce „ostatní“ jsou skripty použité pro ladění
- tul zranitelnost – podrobnosti o zranitelnosti nalezené na tul.cz

Příloha B

Zranitelnost na tul.cz



Obrázek B.1: Zranitelnost nalezená na tul.cz v prohlížeči Aurora